Institute for Computer Graphics and Vision
Graz University of Technology
Graz

Visual Links to Hidden Content

Master's Thesis

Thomas Geymayer, BSc
tomgey@gmail.com

February 2013

Supervision:

Prof. Dr. Dieter Schmalstieg
Dr. Alexander Lex
DI Markus Steinberger
Dr. Marc Streit

Abstract

Visual links are lines drawn on top of an existing visualization to create connections and guidance between related regions. With modern operating systems, information is often distributed across multiple applications. As screen space is limited and applications may overlap, regions containing important information are prone to being invisible to the user.

In this thesis we present two new visualization techniques that help users finding and exploring important information hidden somewhere on the desktop. Visual cues and interaction methods allow for a fast identification and navigation to such hidden content.

Keywords: visual links, off-screen, hidden content, preview pop-up

Zusammenfassung

Visual Links sind Linien, die über eine existierende Visualiserung gezeichnet werden können um zusammengehörige Regionen visuell zu verbinden. Bei der Verwendung moderner Betriebssysteme sind Informationen meist über mehrere Anwendungen verteilt. Durch die beschränkte Größe der Bildschirme und auch durch überlappende Fenster sind Regionen mit gesuchter Information für den Benutzer oft nicht sichtbar.

In dieser Arbeit präsentieren wir zwei neue Ansätze, um dem Benutzer beim Finden verdeckter Inhalte auf dem Desktop zu helfen. Visuelle Hinweise und interaktive Visualisierungen ermöglichen es dem Benutzer, rasch versteckte Informationen zu finden und diese auch anzuzeigen.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Place	Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Ort	Datum	Unterschrift

Contents

Chapter 1

Introduction

In modern information analysis and exploration relevant pieces of information are often spread across large areas of screen space. Additionally, they can also be available in different applications, all visible at the same time. To help the user quickly identifying related information, Waldner *et al.* [WPL+10] have proposed a system which uses *visual links* to connect the related pieces of information distributed among multiple applications.

Although plenty of display space is available on today's computer setups, often some important parts of information are hidden. There are various reasons for this, like minimized or covered application windows and also parts of applications, which can only be seen if scrolling the applications viewport. In this thesis we present a visualization for such hidden regions embedded into a visual links system.

1.1 Problem Analysis

To the knowledge of the author, no such system exists yet. In order to realize an effective hidden content visualization, the following requirements need to be satisfied:

1. **Show location or direction**
   To help the user in building a mental map of the explored documents and applications and to allow identifying pieces of information already inspected, it is needed to show the exact location or at least the approximate direction in which the information can be found.
2. **Show amount of hidden information**
   Often information is accumulated in small areas, whereas little information is spread among the remaining parts of the active documents and applications. A possible strategy for exploring such data sets is to prioritize large build-ups of information over small pieces of individual information. For supporting such a strategy visual clues should indicate the amount of information available at a specific location or in a certain direction.
3. **Guidance to reveal hidden information**
   After the user has selected a piece of hidden information for further exploration, it should be possible to quickly navigate there without requiring much manual interaction. Fast navigation between all available chunks of information is essentially for an efficient exploration.

1.2 Challenges

Aside from addressing the proposed requirements, building an interactive tool to realize a visualization for such *off-screen contents* raises several challenges which need to be met:

1. **Rearranging Windows**
   The arrangement of windows visible on the desktop can change at any time. Windows can be moved around, resized and also minimized or closed. The visual links system not only needs to detect such changes but also needs to react on them and adapt or recalculate all links as required.
   As most operating systems do not allow user applications to be notified about changes in the window arrangement, it is required to monitor the list of opened windows and detect any changes. The system needs to check for changes in geometry as well as whether new windows have been opened or existing windows have been closed. Special care needs to be taken as different operating systems have different ideas of what exactly a window is. For example, if using Windows every opened tool-tip or menu is reported as being a window, whereas Linux just reports conventional windows as being windows.
2. **Changing Content**
   Another source of possibly changed linked regions is the change of actual contents inside any available window. For example, scrolling a web page opened in a browser translates every linked region along the scrolling direction. The user can also open a new web page or browser tab, which requires to completely remove all existing regions found in the old document and search again in the new page.
   To detect changes inside an applications window it is possible to use an image based method and periodically compare captured images of each windows area. This method though would not detect any changes inside hidden regions, which leads to the requirement for every application to report changes in its contents by itself – visible as well as hidden.
3. **Application Integration**
   The visual links system just draws links and does not provide or visualize any other information. Instead, already available applications and visualization frameworks are used. This requires to communicate with those applications, either by creating plug-ins or if that is not possible by directly modifying the application. A protocol is needed to exchange data between the visual links system and all connected applications.
4. **Interactivity**
   Investigating available information requires interacting with the visualization. Visible as well as invisible information changes, and the user wants to get more information on demand or inspect certain areas in detail. Thus the visual links system should not only react on changes but also run at a reasonable speed. With large base representation this can be hard to achieve. To still keep the visualization responsive, the user should be provided with a partial visualization or at least an indication for some work being still in progress.

5. **Multi-platform Support**
   To avoid limiting the choice of operating system by the used technologies, the visual links system should be usable with all major operating systems. Using cross-platform frameworks allows using a single implementation for multiple platforms, though meeting this challenge is not always straight forward, as specific features might not be available on all platforms.

In the following sections we will first have a look at related work and afterwards describe how we address the identified requirements and challenges by extending the ideas of a visual links system with some simple but effective visualizations and interaction methods.

1.3 Contribution

The aim of this thesis is to create a multi-platform implementation of the visual links techniques introduced by Waldner *et al.* [WPL+10] and develop methods to also cope with hidden content. The main contributions are two visualization techniques for covered regions (see Section 3.2.1) and off-screen content (see Section 3.2.2) respectively.

Chapter 2

Related Work

Our research is mainly based on the field of *information visualization*. To assist users in the process of exploring related pieces of information spread across multiple screens, techniques like Focus+Context (F+C) (*cf.* Section 2.1) can be beneficial. Incorporating the principles of Gestalt psychology, *visual links* (*cf.* Section 2.2.1) have been shown to provide an efficient technique for visualizing connections between related pieces of information, also spread across multiple applications.

2.1 Focus+Context

Often one is only interested into certain parts of the whole visualization (Focus) but still wants to be able to perceive the location inside the whole *context*. F+C methods address this in different ways [KHG03]:

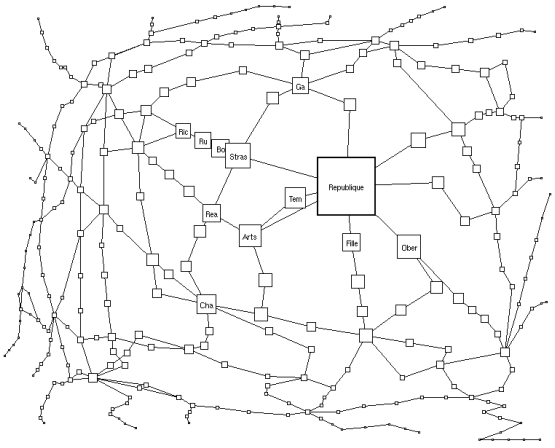
2.1.1 Distortion-Oriented F+C Methods

Probably the most prominent instance of distortion-oriented F+C methods [LA94] is the *fisheye* view. No content is removed but instead the whole view is distorted, such that important parts are magnified while the remaining content is compressed to make space available for the focused region(s).

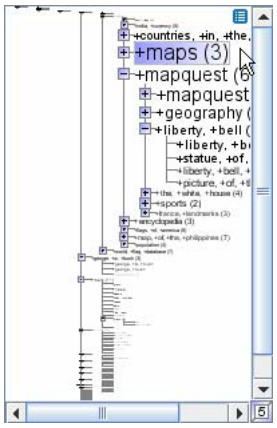
In 1976 Saul Steinberg created a cover image for the *The New Yorker* magazine illustrating a possible view of new yorker inhabitants onto the rest of the world (see Figure 2.1(a)). Analogous to a wide-angle *fisheye* lens, which captures objects around the focal point in high detail, but the rest of the surrounding area strongly distorted with just low details, he has drawn important streets and houses of Manhattan, bounded by the Hudson river whereas on the other riverside, the rest of the United States continues as a quadrilateral shape bounded by Canada, Mexico and the Pacific Ocean and containing an area called Jersey and some other large towns like Los Angeles or Las Vegas. On the other side of the Ocean there are just three regions depicting the existence of China, Japan, and Russia.

Inspired by Steinbergs drawing, George Furnas [Fur81][Fur86] has analyzed several implicitly used natural fisheye strategies and formalized *"generalized fisheye views"* by introducing a *"Degree of Interest"* (DOI) function, which rates each piece of information according to certain criteria. For creating a fisheye view simply the `n` best elements are used, where the DOI is typically calculated using a combination of global importance and distance to the focal point.

(a) *View of the World from 9th Avenue* [Ste76]



(b) Textual Fisheye Tree View [SB94]



(c) Fisheye Tree View [TAvHS06]

Figure 2.1: Fisheye View of a Graph [TAvHS06] and Saul Steinbergs illustration "View of the World from 9th Avenue"[Ste76]

Another visualization having fisheye properties is the *Perspective Wall* developed by Mackinlay *et al.* [MRC91], which shows details on a 2D region, whereas the context is distorted using a 3D projection allowing for a fast implementation through the use of hardware acceleration. The same authors developed also another 3D visualization called *Cone Trees* [RMC91], which allows users to interact and explore hierarchically structured data by moving around the tree using a virtual camera which magnifies nodes in front of the camera leading to a fisheye effect. For large numbers of nodes (more than 1000) though, *Tree-Maps* as described by Johnson and Shneiderman [JS91] are probably the better visualization of choice.

Later Sarkar and Brown [SB94] apply fisheye based techniques to graphs and maps by applying different levels of distortion. This allows the focal node to be shown in high detail, while the other nodes are distorted to provide space for the magnified node (see Figure 2.1(b)). The amount of detail shown for a vertex of the graph and its size are proportional to the distance of each individual node to the center of the imaginary fisheye lens.

More recently Tominski *et al.* [TAvHS06] have created a *Textual Fisheye Tree View* which uses the ideas of a fisheye approach and applies different scales to the vertices of a tree structure similar to the directory tree as used in many operating systems file browsers (see Figure 2.1(c)).

Other examples of distortion based F+C methods are stretched rubber sheets [SSTR93], hyperbolic trees [LRP95] and the *Magic-Eye-View* [KLS00].

2.1.2 Overview Methods

Overview Methods show focus and context in different regions or windows not directly interconnected. Usually an overview of the whole visualized content is displayed in a small

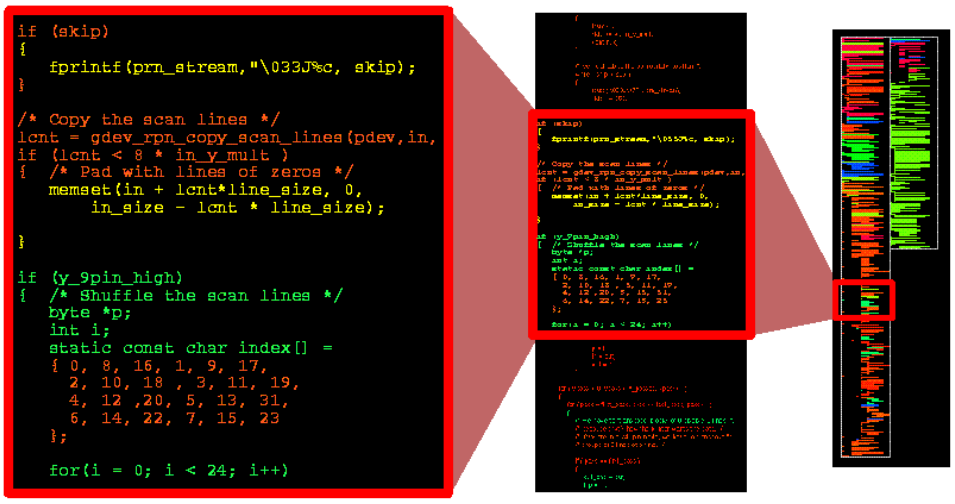


Figure 2.2: Browsing code or other textual data at different levels of detail. From left to right the same focus area is shown inside an increasing large context whereby the first the font size is reduced and afterwards the text replaced by simple lines. Different color denote the date a line has been modified last. [BE96]

window, whereas the small focus area is displayed magnified into a larger window. Often the viewport of the focus area is highlighted in the overview window. This type of visualization is quite common for document and image viewers but also used by many file browsers which show an overview of the directory hierarchy in one panel and more details and contents of a single directory in another panel. Another application area are digital cameras, which typically show an outline of the whole image area while zooming into an image to keep track of the current location.

The same idea is used also to visualize large amounts of textual data like code, by using smaller fonts for intermediate zoom levels, and replacing text by simple straight, geometric lines for low levels of detail. This allows visualizing a lot of code at the same time, while still preserving structure like indentations and empty lines.

An early example of such a tool is *Seesoft* developed by Eick *et al.* [ESS92], which can simultaneously cope with up to 50.000 lines of code showing files in columns and the individual lines as rows. Dependent on the current task, the color of each row is determined by one of several statistical properties, like for example the date of last change of each individual line.

In 1996 Ball and Eick [BE96] described different approaches for visualizing code structure. Besides a *Line Representation* (see Figure 2.2) similar to the one used in Seesoft, they even went a step further and reduced each line to a single pixel, which allows them to analyze over a million lines of code at once.

Another similar visualization has been used by Eagan *et al.* [EHJS01] to assist in finding faults in software. They use a line representation of the code where different colors encode how often lines have been executed during failed, succeeded and both types of tests respectively.

2.1.3 Filtering

In photography filters are used to enhance or alter the way pictures are captured. The same idea is imitated by filtering F+C methods where the user can place an arbitrary shape resembling a filter over the visualization and provide more detailed or otherwise enhanced information inside the area covered by the filter.

Magic Lenses are introduced as a *see-through interface* by Bier *et al.* [BSP+93]. They are examples of F+C filters with 2d shapes. Extending the idea to the third dimension, three-dimensional lenses have also been successfully used [VCWP96].

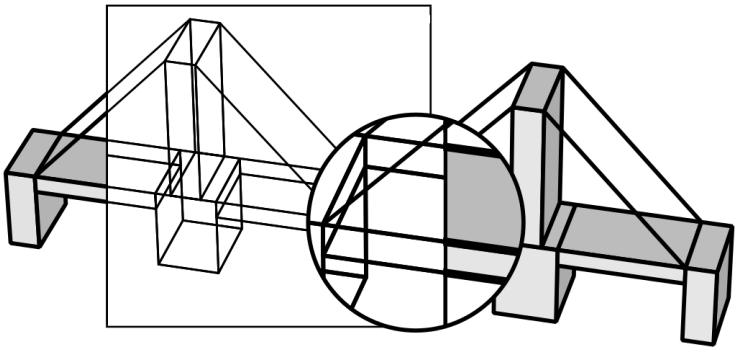


Figure 2.3: Two 2d Magic Lenses applied to a 3d model of a bridge. A rectangular lens reveals a wire-frame model and a circular lens magnifies its area. Within the intersection of the lenses, both filters are applied simultaneously. [BSP+93]

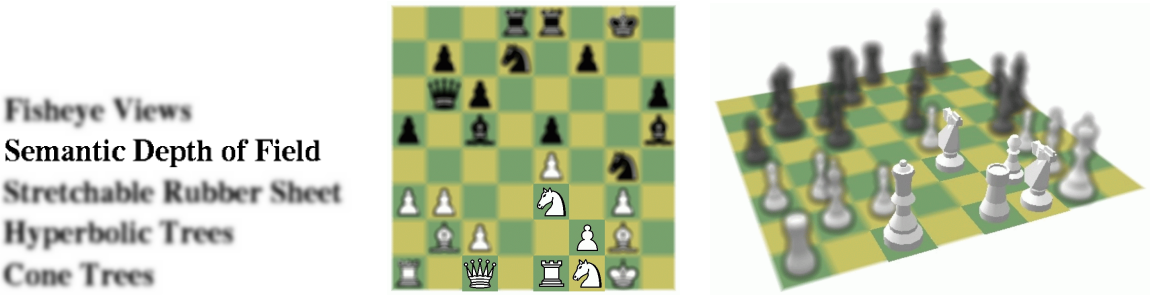
A more recent filtering F+C method is the *ClearView* lens developed by Krüger and Fogal [KF09], which applies a special rendering mode to volumetric data to allow the focal parts getting clearly visible even while being covered by surrounding context.

2.1.4 In-Place F+C

Instead of distorting or changing the amount of visibleinformation, one can also modify the way it is visualized. Such *in-place F+C* methods are sometimes categorized as *cue-based techniques*[CKB09]. Generating visual cues is achieved for example by changing colors or applying blur to the base representation [ZWSK97].

In optics the *Depth Of Field* (DOF) describes a distance interval between where all objects appear sharp in a captured image. This effect is often used in photography and cinematography to direct the users attention to a certain object or area which is inside the focal area of the used lens. The same idea has been used by Kosara *et al.* [KMH01][KMH+02] to create a F+C technique called *Semantic Depth of Field* (SDOF). Instead of using the depth of objects they use semantic properties to discriminate important object from less important context information. After this classification unimportant objects are blurred and combined with the focal objects left unchanged. The SDOF can be applied to textual data as well as two- or three-dimensional environments as shown in Figure 2.4.

A similar method has later been developed by Khan *et al.* [KMFK05]. Instead of blurring they darken the whole screen and only leave a circular region around the focal point



(a) Text (b) Chess 2d (c) Chess 3d

Figure 2.4: Semanitc Depth of Field [KMH01](SDOF) used to highlight text areas 2.4(a) and chessmen on a two- 2.4(b) or three- 2.4(c) dimensional chessboard.

unchanged. The idea behind this visualization is an analogy to the spotlights used in theatric productions to direct the audiences attention. A user study has shown that especially on large displays their spotlight technique leads to a large gain in performance if compared to simply using the cursor for pointing the audience to a certain position.

2.2 Gestalt Connectedness

Initially postulated by Wertheimer in 1923, the *Gestalt principles*[Wer23] incorporate mental laws on how humans perceive scenes they can see. For example, objects of a similar color, shape or size are usually perceived as forming a group, which often leads to perceiving more information then the sum of the individual objects. Objects being in close proximity or connected by lines are also instances of the grouping principles.

Several user studies have shown that connectedness is a very strong grouping principle [PR94][ZK10], often outperforming other grouping principles like similarity or proximity. Especially in a cluttered environment, searching for highlighted objects, which are not connected, can lead to a severly degraded search performance, as one has to perform a *serial search*[TG80]. This requires scanning through the whole visible area and looking for relevant elements without missing any of them. Motivated by the Gestalt principles, many visualizations connecting elements using lines or enclosing shapes have been developed.

2.2.1 Visual Linking

To reduce distances one has to move around in a pen- or touch-controlled environment, Baudisch *et al.* [BCR+03] have an alternative Drag-and-Drop technique called *Drag-and-Pop*, which upon dragging a file around moves all launch icons for compatible applications towards the dragged file while still maintaining a connection to their original locations using a rubber band (see Figure 2.5).

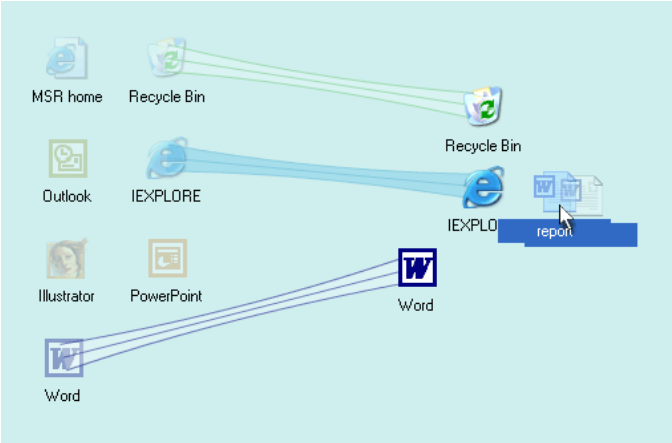
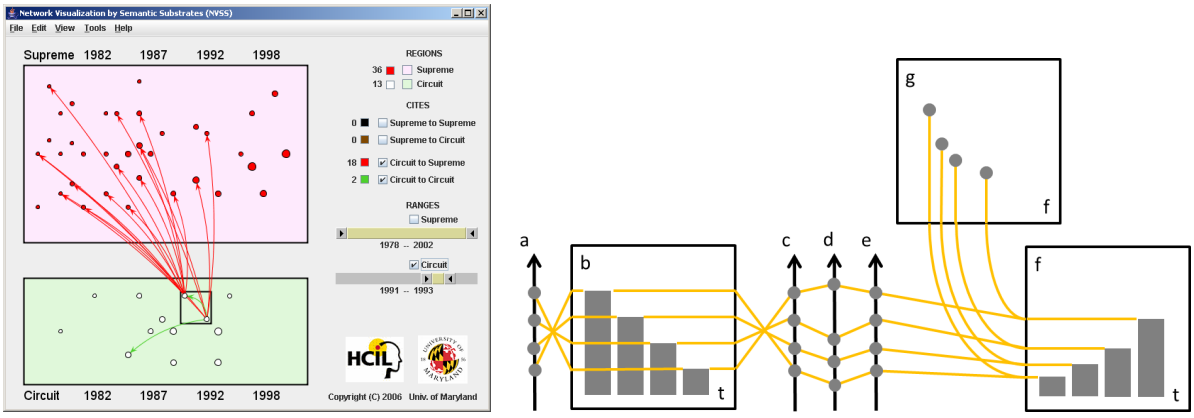


Figure 2.5: *Drag-and-Pop* [BCR+03]: The Word file can either be deleted by moving it over the recycle bin or opened by dragging it onto one of the two supported applications.

Semantic Substrates [SA06][AS07] categorize nodes of a network based on the values of certain attributes and arrange them in different regions. Afterwards the user dynamically modifies some selection criteria where the relations between matching nodes are visualized by drawing links (see Figure 2.6(a)).



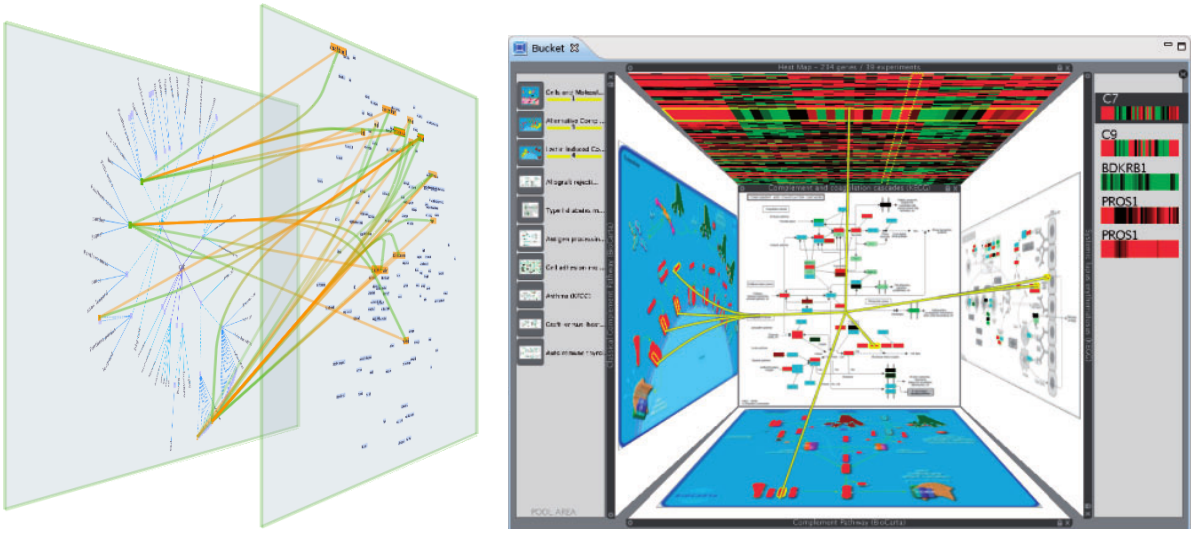
(a) Semantic Substrates [SA06] (b) ConnectedCharts [VM12]

Figure 2.6: 2D Links

ConnectedCharts uses a very similar layout of links recently proposed by Viau and McGuffin [VM12] to connect data entries occurring in multiple charts (see Figure 2.6(b)).

Still showing relations between multiple views of the same or related data in 2d visualizations, Collins and Carpendale with their *VisLink* [CC07] technique and Streit *et al.* with their *connections between pathways* [SLK+09] project the separate data views into a three dimensional space and draw links between selected items to allow the user quickly exploring the relations among multiple views (see Figure 2.7).

Bubble Sets, another way of visually connecting objects, have been proposed by Collins *et al.* [CPC09] to show on top of existing visualizations the relations of elements belonging



(a) VisLink [CC07] (b) Caleydo Bucket [SLK+09]

Figure 2.7: 3D Visual Links

to different sets. Instead of links they use bubble like shapes, each enclosing all elements belonging to a specific set.

2.2.2 Edge Bundling

Without taking any actions a visualization can quickly get cluttered while showing too much information with just limited space available. Several techniques are being used for reducing such visual clutter [ED07]. As it is not feasible to remove or move any nodes of the visual link graph, we will focus on techniques trying to reduce clutter by optimizing the way edges between linked nodes are drawn. Edges are regarded as being flexible lines similar to electrical wires which are then bundled together at parts of their routes where they are installed in close proximity.

With *confluent diagrams* Dickerson *et al.* [DEGM04] draw non-planar graphs with possibly many crossings in a crossing-free manner. This is achieved by combining crossing edges into a common edge bundle and let them fan-out towards the nodes they are connected to.

Flow maps [PXY+05] visualize hierarchical flows by merging flows along common routes into a single larger flow. The idea of *edge bundling* is also used by Holten [Hol06] to visualize relations inside a set of hierarchical data using a hierarchical tree layout and applying an edge bundling algorithm to it (see Figure 2.8).

For general graphs *Geometry-Based Edge Clustering* can be used. Inspired by the way road maps are drawn Cui *et al.* [CZQ+08] developed a method which first creates a control mesh based on the distribution of edges and afterwards uses this superior structure to bias the edge bundling process.

As using geometry-based edge clustering often leads to edge bundles hard to follow [HVW09],

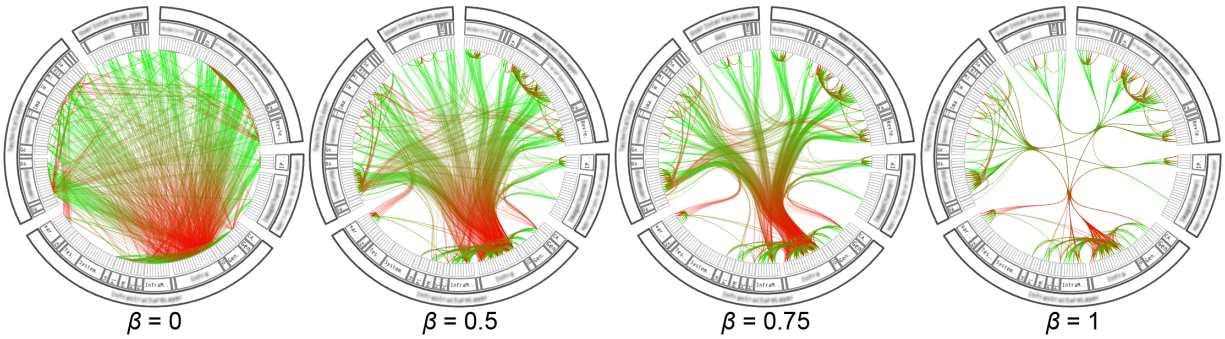


Figure 2.8: *Drag-and-Pop* [BCR+03]: The Word file can either be deleted by moving it over the recycle bin or opened by dragging it onto one of the two supported applications.

Holten *et al.* introduced a *Force-Directed Edge Bundling* algorithm [HVW09] for visualizing general graphs. They do not rely on creating a control mesh, but instead subdivide the underlying straight-line node-link graph and simulate applying spring and electrostatic forces to the subdivision points. An iterative approach is used to find an approximate solution within feasible time.

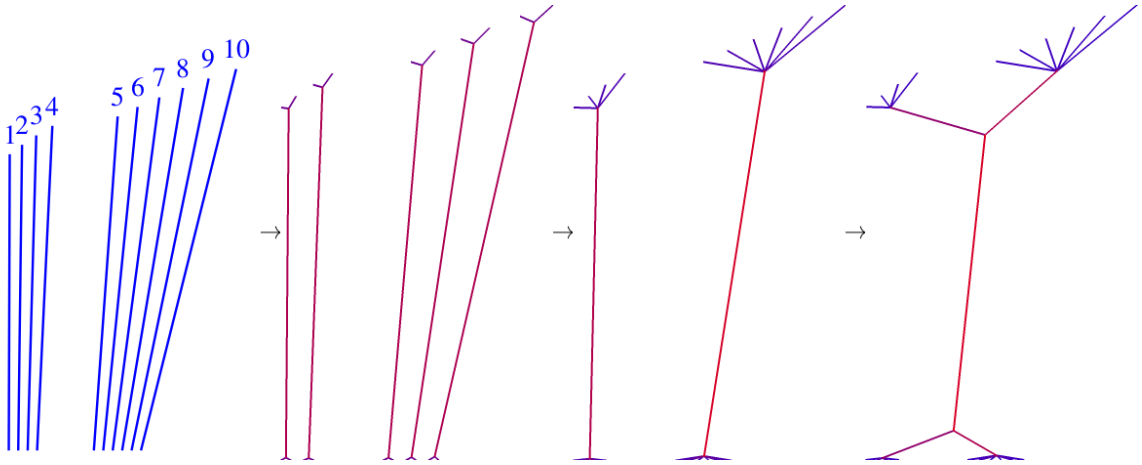


Figure 2.9: Multilevel agglomerative bundling [GHNS11]: Iteratively applying the bundling algorithm results in an increasing number of edges being merged together into bundles.

Recently Gansner *et al.* [GHNS11] presented an approach inspired by the way a human would bundle electrical wires. In an iterative process lines with similar routes are identified and grouped together (see Figure 2.9), either be adding to an existing bundles or by forming new ones. The objective function of this optimization problem is the sum of the lengths of all edges, whereas a group of bundled edges only counts once. Additional constraints like limiting the turning angles may be added to get a more visual pleasant result.

2.2.3 Visual Links across applications

For investigating into different aspects of analyzed data, often multiple applications are used, each providing a highly specialized visualization technique. To increase the efficiency of such an analyzing strategy North and Shneiderman have developed *Snap-Together Visualization* [NS00], an API for coordinating and synchronizing data and highlights across multiple visualization applications.

As the human eye has a very small field of view [War04] but currently used screens are often very large, information especially in peripheral regions is easily overlooked [HS04][BB09]. Especially if multiple regions of interest exist, very strong visual cues are needed to direct the users attention.

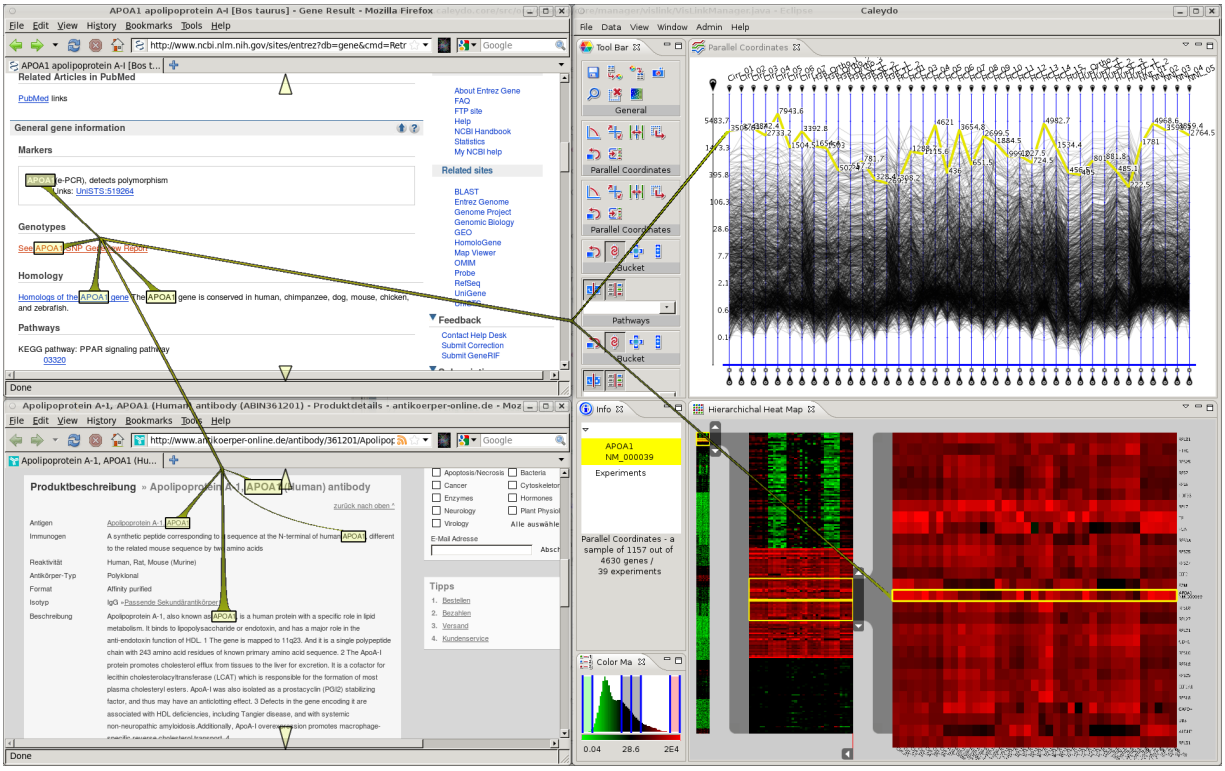
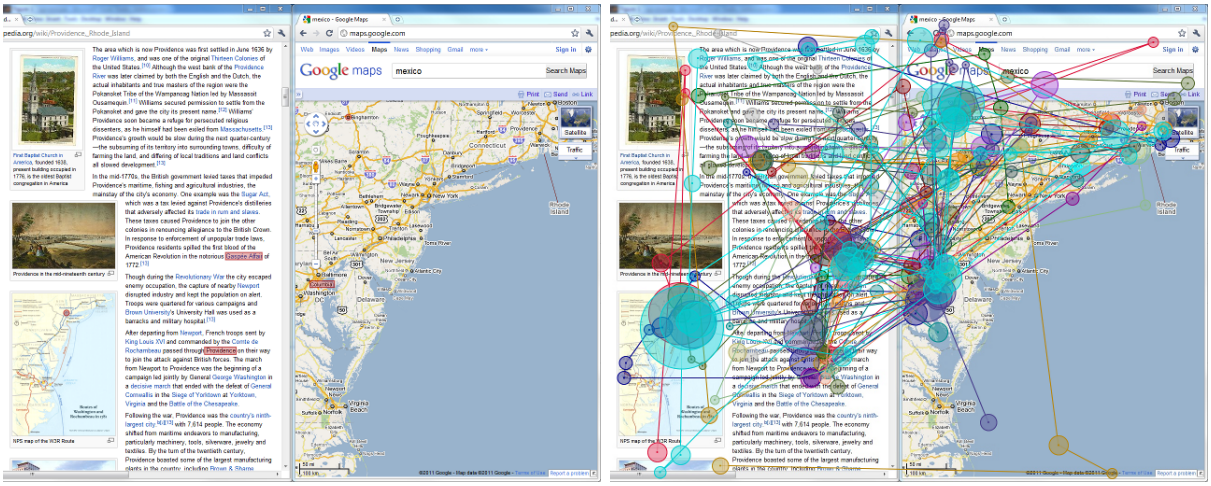


Figure 2.10: Exploring results of a biomedical experiment connecting regions of interest spread among multiple applications using visual links. [WPL+10]

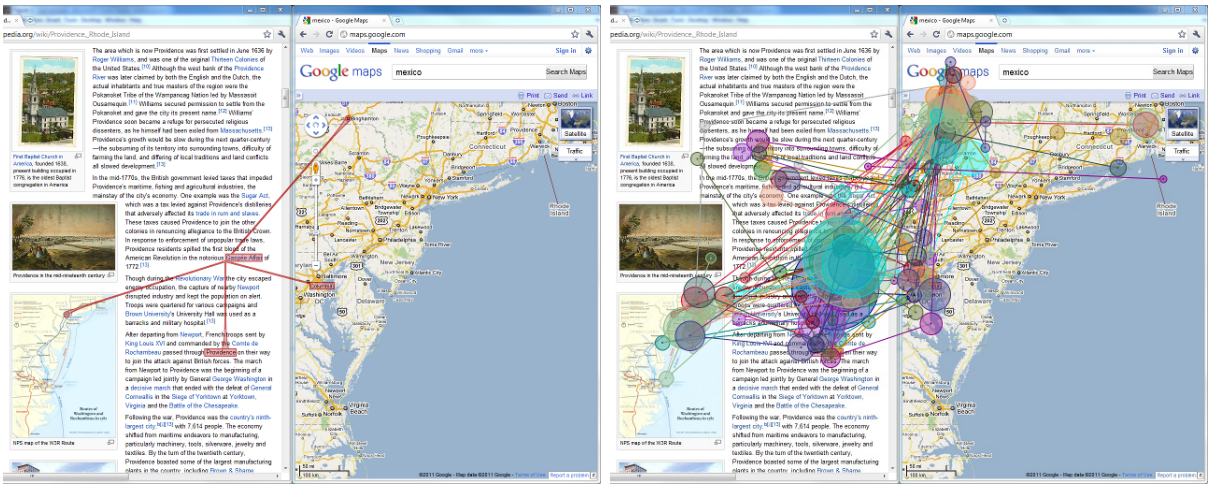
Using connectedness as strong visual cue, Waldner *et al.* [WPL+10] have created *Visual links across applications*, a program visualizing relations between locations of interests spread around multiple applications. Applications can connect to a background process and report regions of interest and an associated identifier, either initiated by the user or automatically. All connected applications receive the identifier and can in turn report matching regions on their own. After collecting all regions they are connected by visual links drawn on top of the desktop as shown in Figure 2.10.

Steinberger *et al.* [SWS+11] have conducted a user study to show the improved performance of visual links. Several users were instructed to count highlighted regions, either just

highlighted by frames drawn around them or additionally also connected with visual links. Results confirmed the expected performance gain. Using an eye tracker the search strategy of the users has been analyzed. With simple highlights users need to scan through large parts of the image to find all regions (see Figure 2.11(a)) whereas visual links strongly guide the user towards all regions by just following the lines as can be clearly seen in Figure 2.11(b).



(a) Highlight



(b) Visual Links

Figure 2.11: Gaze plots showing how users scan through an image to count highlighted regions. Simple highlights (a) are compared to straight visual links (b). Comparing the result shows clearly the strong guidance effect of visual links. [SWS+11]

Encouraged by the performance of visual links they have been further improved by Waldner *et al.* to extend the links also across multiple computers operated by multiple users at the same time [WS11]. As simple straight links are vulnerable to obscure important information, Steinberger *et al.* [SWS+11] also take the base representation into account for creating optimal link routes preserving as much information as possible (see Figure 2.12).

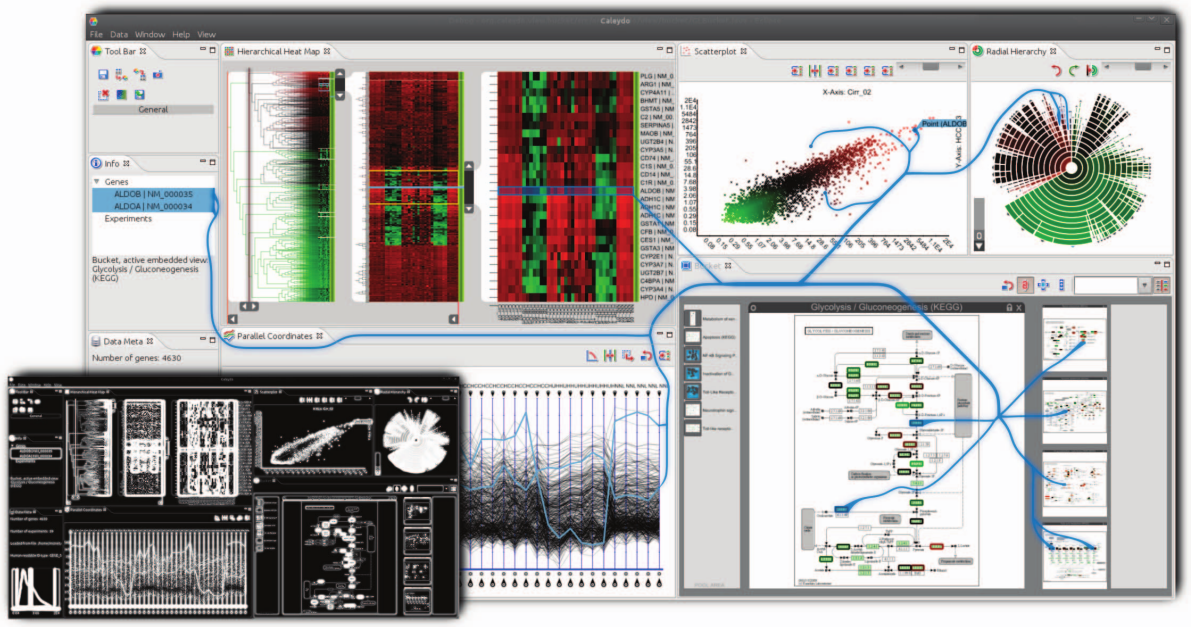


Figure 2.12: Context-Preserving visual links try to avoid routing links across salient regions. [SWS+11]

2.3 Hidden Content

Only a limited amount of content can be shown to the user at the same time. Consequently always some content is not visible to the user. A trivial reason for content being invisible is a document available somewhere, but currently not opened in any program. If content is available inside an application on the desktop, it may still not be visible to the user.

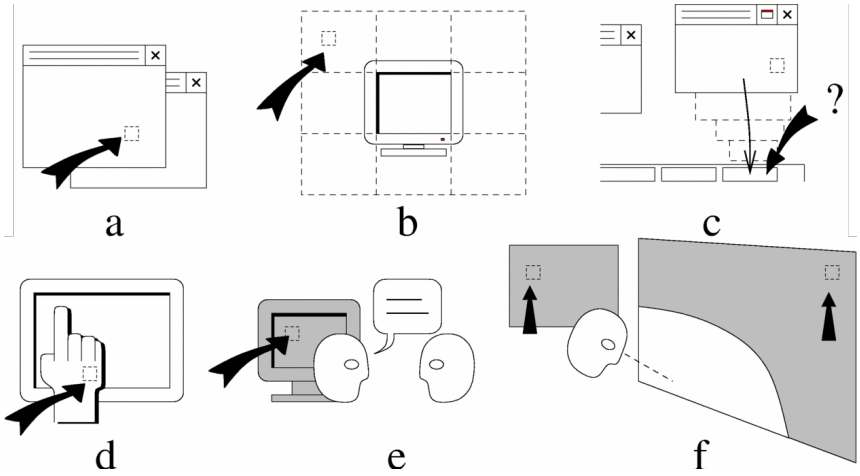


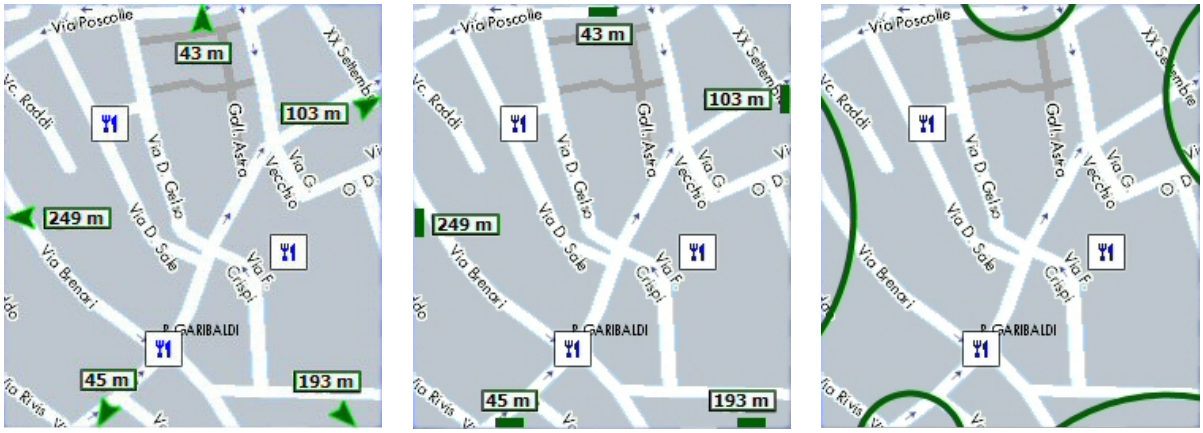
Figure 2.13: Users can not see content covered by another window (a), outside a virtual viewport (b) or inside a minimized application (c). If content is available on the screen it may still be invisible to the user due to physical occlusion (c) or being outside the focus of attention (d)(e) [BDB06].

According to Berzerianos *et al.* [BDB06] two main reasons cause content being invisible to the user. Either content is not displayed, as for example it is covered by another window, or the user does not see it, because his attention is focused somewhere else (see Figure 2.13). In this thesis we concentrate on off-screen content – content available on the desktop but currently not displayed.

While relating pieces of information using visual links, little attention has been spent on how to visualize information currently not visible, for example due to being covered by other windows or scrolled out of the visible viewport of an application. Waldner *et al.* [WPL+10] indicate information scrolled outside a windows region by drawing arrows pointing into the according directions. Information hidden by overlapping windows is not considered at all, but was mentioned as being important for future research.

2.3.1 Off-screen Content

In the context of small screens like found on mobile devices, several techniques have been proposed to indicate the locations of points of interest on pan- and zoom-able maps [BCG06]. As the typical task in such an environment is to exactly locate points, usually direction and distance are encoded in the visualization. A well know approach consist of arrows pointing into the direction of hidden positions possibly encoding the direction by scaling or simply adding a textual label.



(a) Arrows (b) City Lights [ZMG+03] (c) Halo [BR03]

Figure 2.14: Using arrows (a), *City Lights* (b) and *Halos* (c) to visualize distance and direction to locations on a map outside the visible screen space. [BCG06]

Using bars at the borders of window regions, *city lights* [ZMG+03] indicate direction and width or height of hidden regions. Encoding distance and directions instead, Baudisch *et al.* have developed a technique called *Halo* [BR03] where circles are drawn around the hidden locations. The size of these circles are chosen such that they are just large enough to intersect with the visible region of the screen. Now the curvature of the resulting arcs encodes both distance and direction.

Figure 2.15: *Wedges* use culled triangles to encode direction and distance to offscreen locations. Compared to *Halos* (as shown on the small image) Wedges can create much better cues guiding to the hidden locations. [GBGI08]

To overcome visual clutter possibly occurring with methods like city lights, Gustafson *et al.* [GBGI08] use *wedges* to encode direction and distance. Instead of circles, isosceles triangles are placed, such that the apex is at the location of the hidden object, and the triangle intersects with the visible region.

2.4 Discussion

F+C techniques are used to emphasize a focal region while still showing some or all contextual information around. Methods like fisheye views [Fur86] distort the image to enlarge the focal region whereas other methods just highlight the focus or darken the context [ZWSK97].

Visual Links have been used for creating strong visual cues between related pieces of information inside and across applications by Waldner *et al.* [WPL+10][WS11] and Steinberger *et al.* [SWS+11]. To reduce visual clutter introduced by drawing visual links, individual links are bundled together similar to edge bundling already applied to general graphs and trees. The discussed visual links systems use a hierarchical bundling algorithm similar to the multilevel agglomerative bundling algorithm of Gansner *et al.* [GHNS11]. For bundling visual links the algorithm can be simplified because the links always appear in a known hierarchic structure.

Information not visible to the user but still present on the desktop has hardly been considered, but identified as fruitful direction for future research [ET08]. Regions covered by other windows are completely ignored by all implementations of visual links. Due to the limited display space available on mobile devices research has been done to investigate on how to guide the users to locations on a map outside the display. Different visualizations like simple arrows [BCG06], city lights [ZMG+03] or wedges [GBGI08] are used to encode distance and direction to locations outside the viewport.

Chapter 3

Concept of Visual Links to Hidden Regions

Similar to the architecture of the system presented by Waldner *et al.* in "Visual Links across Applications" [WPL+10], our prototype consists of a central `Visual Links Server` which communicates with different client applications (see Figure 3.1). Each client application needs to register itself to the server and can afterwards send linking requests to the server. Clients can offer arbitrary methods to their users for creating a new linking request. For example, the user can highlight a word or enter a word into a textbox and use it as an identifier for starting a new linking process.

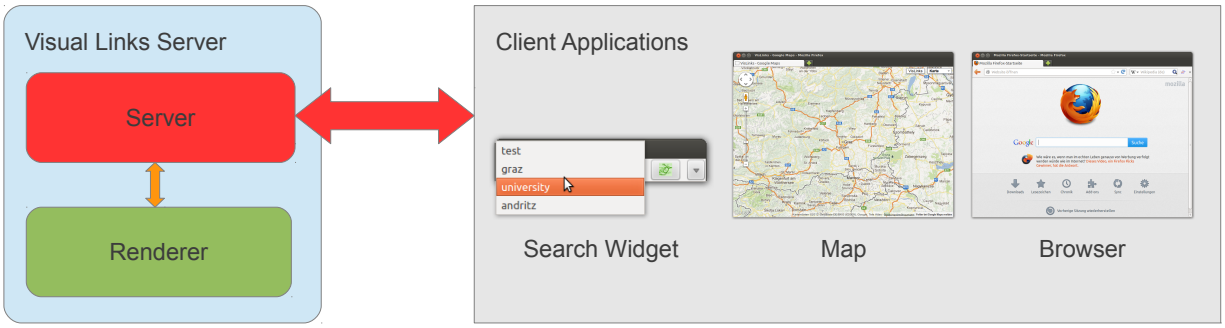


Figure 3.1: Basic architecture of the visual links system. The visual links server communicates with different client applications and creates visual links according to the received data.

After the server has received a linking request, it forwards it to every connected client, enabling each application to add links to information inside their viewport. Upon receiving a linking request, a client should search its contents for instances of the requested identifier and report back the bounding boxes of each found occurrence. For simple selection types like single words, bounding boxes already provide an accurate approximation of the relevant region. To highlight and link more complex shapes, a client is free to use any arbitrary shaped polygon for representing its regions.

3.1 Visual Links

Every client containing information relevant to the user, sends a list of polygons – each enclosing a piece of information – to the server. The visual links renderer now draws the outlines of the polygons on top of the desktop, using a different color for each group of regions belonging to the same linking identifier. To make the highlights more prominent,

the polygons are filled with a slightly transparent variant of the outline color. Afterwards the individual regions are connected using visual links.

Figure 3.2 shows our basic visual links system being used to connect various locations of information between different types of applications.

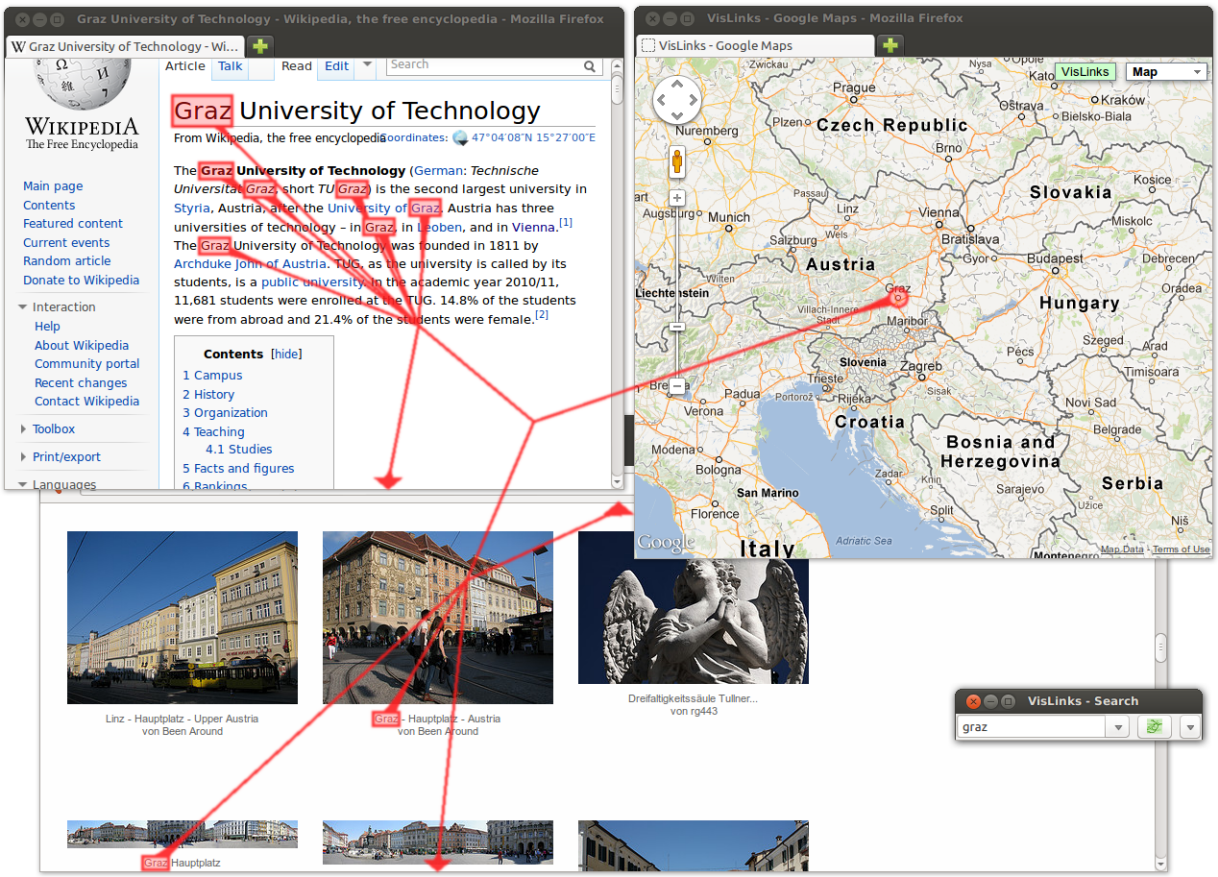


Figure 3.2: Using our Visual Links system connecting information relating to the geographic location of our university (City of Graz) occurring in multiple browser windows and a Google Maps mash-up. The used visualization is basically the same as presented by Waldner *et al.* [WPL+10].

3.1.1 Region highlights

The browser add-on, we will describe in more detail in Section 4.3.1, searches for individual words or text passages matching the requested link identifier, and reports their bounding boxes (see Figure 3.3(a)) to the server.

Our maps mash-up, we will also describe later in Section 4.3.2, can be used to create highlights on maps. It reports circles centered around geographic locations of interest (see Figure 3.3(b)), like for example, cities, districts or countries, to the visual links server.

If clients report regions, which are currently invisible but can be reached through scrolling, these regions are not drawn. Instead an arrow is drawn, pointing towards the next edge of the applications viewport, into the direction of the target (see Figure 3.3(c)). We will discuss the visualization of regions outside a viewport in more detail in Section 3.2.2.

(a) Text (b) Location (c) Scroll

Figure 3.3: Symbology for highlighting text (a), geographic locations (b) and directions to occurrences outside the viewport of a scrollable region (c)

3.1.2 Link Bundling

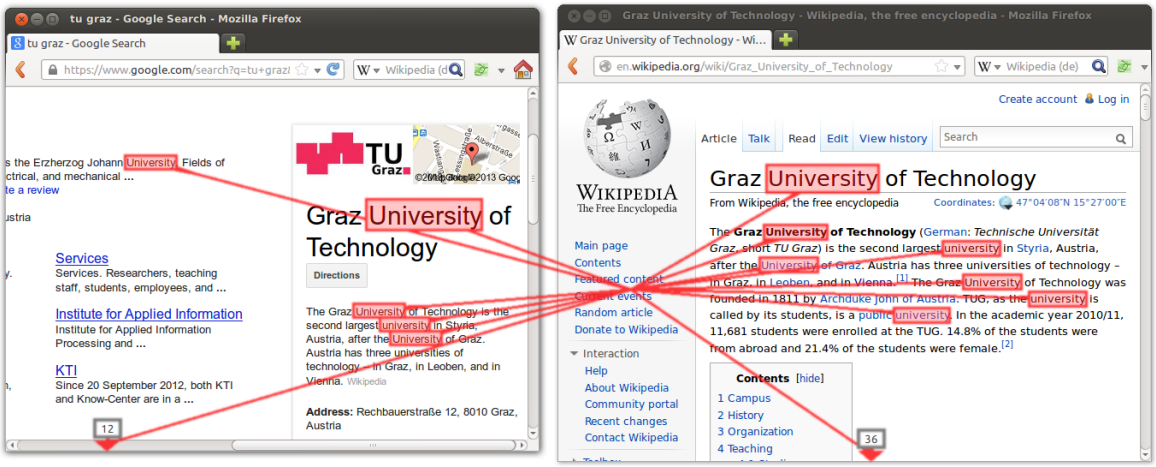
Motivated by the strong visual guidance of *Gestalt connectedness* (*cf.* Section 2.2), the individual region highlights are connected using straight visual links. Connecting all regions to a common center, as shown in Figure 3.4(a), results in a cluttered visualization with much display space occluded by the visual links.

For this reason we bundle together links with proximate routes, using an iterative algorithm – similar to the *Multilevel agglomerative edge bundling* by Gansner *et al.* [GHNS11]. In contrast to multilevel agglomerative edge bundling, we do not construct an edge proximity graph, but instead use the hierarchy created by the fact, that each region belongs to a certain application. First all links are bundled together at the individual centers of each application, and afterwards connected to the common center of all participating applications. A possible visualization using the described method can be seen in Figure 3.4(b). To reduce clutter and keep the total length of all links low, we bias the centers of each separate application towards the common center.

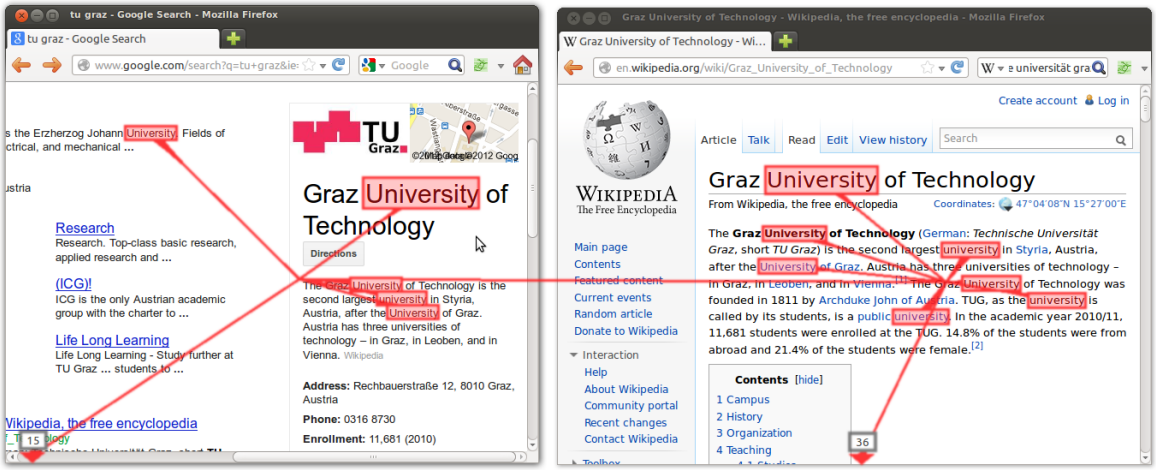
Combining this principles results in the following steps, required to calculate the bundle points:

1. **Client bundle points** The *bundle points* for each client application are calculated by taking a weighted average of all regions belonging to a single application. Visible regions have a much higher influence than hidden regions.
2. **Global bundle point** The geometric center of all client bundle points is calculated and used as global bundle point.
3. **Biasing client bundle points** Every client bundle point is moved into the direction of the global bundle point (see Figure 3.4(c) for an example of two bundle points being biased towards the common center).

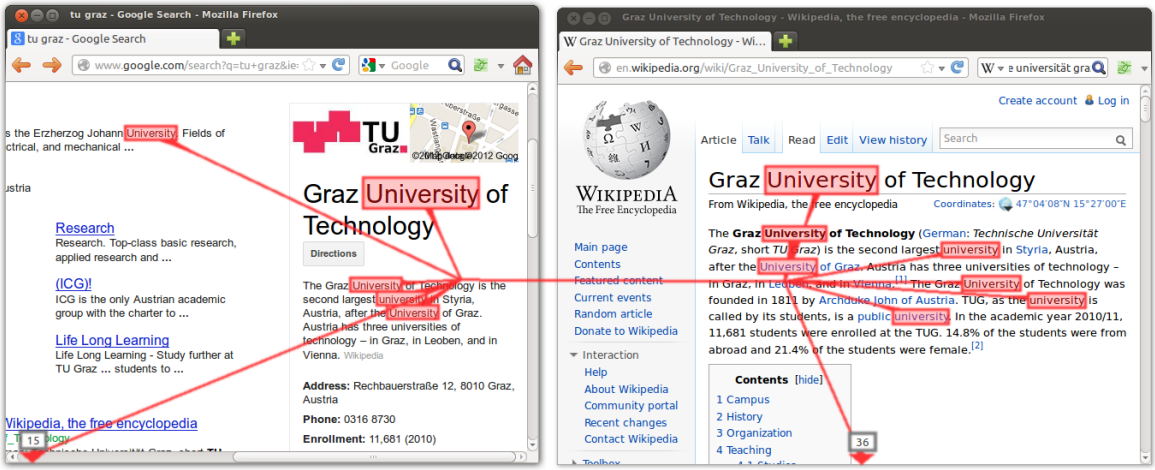
Afterwards straight lines are draw from the common center to each bundle point, continued by forking a link to every region, and finally ending in a smooth transition to the regions highlight.



(a) No bundling



(b) Per application bundling



(c) Per application bundling with center bias

Figure 3.4: Connecting regions with visual links connected to a central point (a), using per application bundle points (b) and with biasing the bundle points towards the center (c).

3.1.3 Link-Region Transition

Figure 3.5 shows how a transition from a link to a connected region is drawn. First the link is expanded to smoothly fade into the region (see Figure 3.5(a)) and afterwards the inner part of the region is cleared to not occlude the information being highlighted (see Figure 3.5(b)).

(a) Expand link (b) Clear region (c) Gaussian blur

Figure 3.5: A transition from a link to a highlighted region. First the link is expanded (a) and afterwards the inner part of the the region is cleared (b). Finally a Gaussian blur filter is used to further smooth the whole visualization (c).

All rendering output is directed to an off-screen buffer first. After everything has been drawn, a Gaussian blur filter is applied to the whole buffer image to further smooth links, highlights and transitions (see Figure 3.5(c)).

Finally, to make the visualization visible to the user, the contents of the off-screen buffer are copied on top of the desktop. Regions on the screen not covered by the visualization are marked with a mask, to allow mouse events passing through. This enables the user interacting with all parts of currently visible applications not covered by our visualization.

3.2 Hidden information

There exist several reasons for information being hidden (*cf.* Section 2.3). In this thesis we concentrate on information covered by other windows on the desktop, or outside the visible part of a viewport. To address the requirements identified in Section 1.1, we use two main visualization techniques helping the user efficiently exploring the available information – an X-ray visualization for covered regions, and preview pop-ups for regions outside a viewport.

3.2.1 Covered regions

Todays desktop environments allow users to run multiple applications concurrently and position and resize the applications windows in any imaginable combination. On the one hand users get the freedom to arrange the used applications according to the individual preferences, but on the other hand it vastly increases the chance of regions showing required information being covered by windows of other applications. To show the user where such

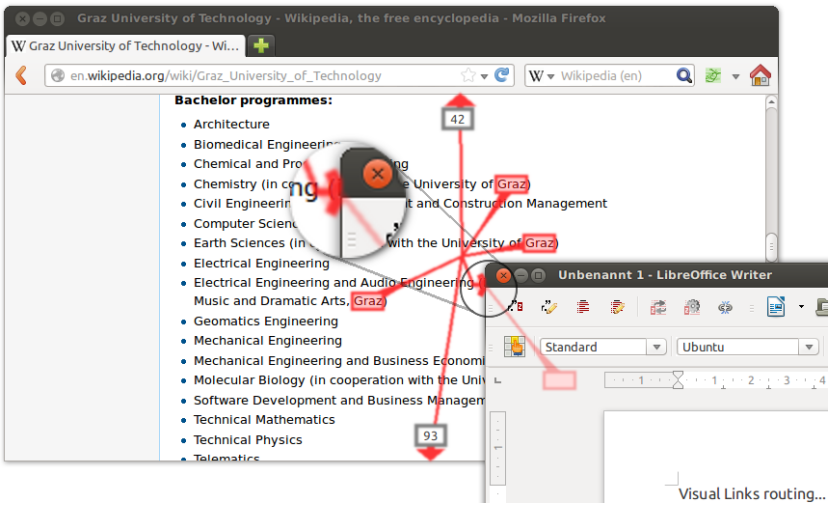


Figure 3.6: We have opened a text editor which hides one occurrence of ”Graz” inside the currently browsed document. The hidden region and its link are drawn using a reduced intensity and opacity color. At the border of the covering window a tunnel portal like icon shows the link is going to continue below the text editor.

hidden regions are located we use a simple version of an X-ray visualization, a technique commonly used in augmented reality systems to make hidden objects visible. We draw the highlight of the region and its connecting link the same way as if the covered region would be visible, but with reduced intensity and opacity of the used color to depict it is not. To intensify the perception of a link being covered by a window, we show an icon symbolizing a tunnel portal (see Figure 3.6) where the link pretends to continue inside a ”tunnel” somewhere below the topmost window.

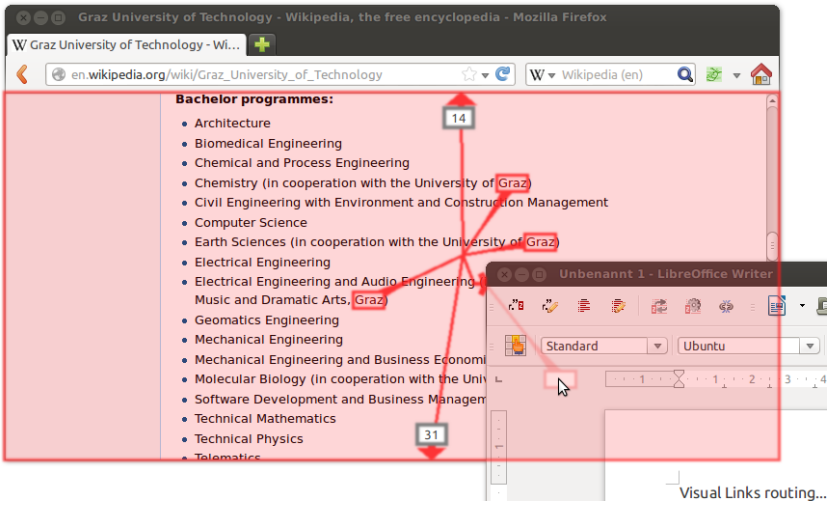


Figure 3.7: Hovering with the mouse above a hidden region triggers rendering an outline of the window the region belongs to.

As all covered regions are still visible to the user but only visualized using a different style,

our visualization satisfies Requirements 1 and 2 as postulated in Section 1.1. To address the last requirement - guiding the user to the hidden regions - a click action with the mouse on the hidden region moves the window containing the very same region to the top of the window stack, which effectively reveals the hidden information. Additionally, if just hovering with the mouse above a hidden region, we show an outline of the window this region belongs to (see Figure 3.7). This helps the user in recognizing which window is about to become visible upon a click, which can get hard to realize at a glance with complex window arrangements.

3.2.2 Regions outside a viewport

Most applications available on todays desktop systems provide the user access to much more information at the same time than there is display space available on the monitors. To still be able to view, for example, large documents in a text editor or whole web pages in a browser, just a small excerpt of the available contents is shown. Such a *viewport* can be moved around through scrolling vertically or horizontally. If searching for information in such applications large parts of the viewed document are usually outside the viewport and therefore not visible to the user. To address Requirements 1 and 2 – showing the the user direction and number of potentially relevant parts inside a document – we use simple arrows pointing into the according direction (see Figure 3.3(c)) and place text labels showing the number of interesting regions in their direction (see Figure 3.8) next to them.

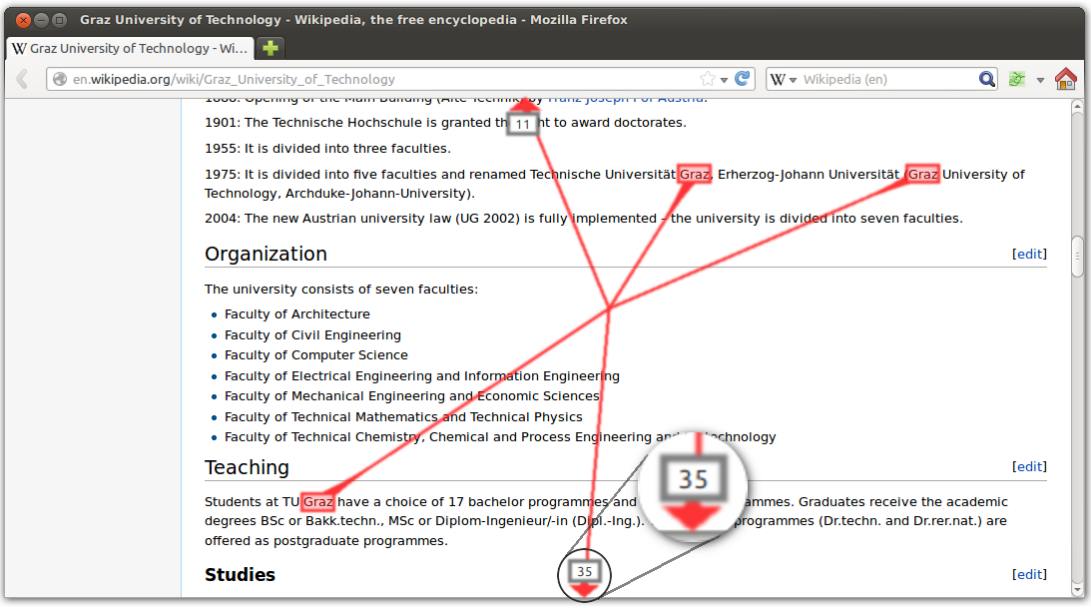


Figure 3.8: Next to the basic scroll indicator as shown in Figure 3.3(c), we additionally place a text box indicating the number of hidden occurrences located in the direction of the arrow.

To not cover too much additional display space, no more than four arrows – one for each edge – per viewport are shown at the same time. All hidden regions are assigned to one

of the four edges, and for each of the resulting groups of regions, the center of gravity is calculated. Afterwards, for each group an arrow is drawn inside the applications viewport, as close as possible to the corresponding center of gravity.

3.2.2.1 Preview Pop-Up

As scrollable regions often exceed display dimensions by far, we can not use a see-through visualization analogous to the visualization of regions covered by other windows (*cf.* Section 3.2.1). Most of the time the highlights of hidden regions would be located outside the desktops area. Instead, upon hovering with the mouse above the text label showing the number of hidden regions (as described in the previous Section 3.2.2), we show a pop-up window (see Figure 3.9(a)). This pop-up window displays a preview of the whole scrollable area and can be navigated using keyboard shortcuts or mouse gestures. The user can zoom and pan the view to explore all available content (see Figure 3.9(b)). Regions of interest are highlighted using simple bounding boxes. To make the bounding boxes more prominent, especially if zoomed out of the preview, the width of the lines used to draw them does not change with the zoom level.

(a) Preview pop-up (b) Zoom/Scroll

Figure 3.9: The preview pop-up (a) displays an outline of the whole document with all occurrences of the requested information being highlighted. The large red rectangle visualizes the location of the viewport inside the actual application. Once zooming into a more detailed view (b), regions can be selected with higher precision. The orange scrollbar at the right side of the window indicates the current vertical location inside the whole applications area.

For a fast correlation of the preview window and the actual document, the current viewport of the application is highlighted by an accordingly sized rectangle. Increasing the zoom level again hides some parts of the preview image. To indicate this fact scrollbars depict

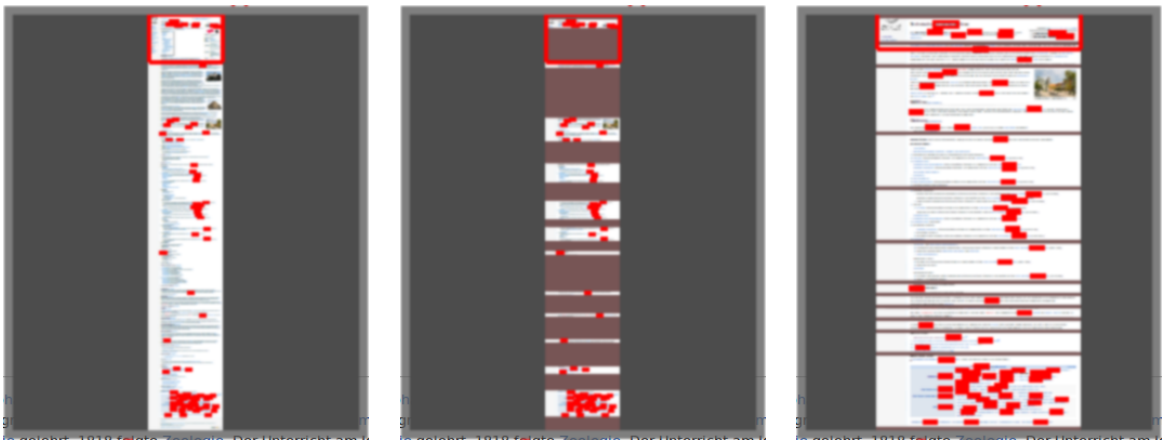
the current location inside the whole preview image. Once an interesting region has been identified by the user, a mouse click at that location hides the pop-up window and scrolls the document to the location of the requested information.

Partitioning

Most of the time, users are only interested in certain parts of documents containing relevant information. Similar to the fisheye techniques discussed in Section 2.1.1 we compress unimportant regions to gain space for the relevant parts. As documents are typically much higher than wide, we only remove rectangular regions of the same width than the document.

To decide which regions should be considered unimportant we perform the following steps, while walking vertically through the document:

1. Calculate bounding boxes of all region highlights.
2. Loop through all bounding boxes and mark region with a certain margin above and below as important (see Figure 3.11).
3. Hide or compress all unmarked and therefore unimportant regions as shown in Figures 3.10(b) and 3.10(c).



(a) Preview pop-up (b) Partitions (c) Compressed

Figure 3.10: Showing a whole web page at once inside a preview pop-up often requires scaling the image down to a size where it is not possible to detect much information any more (a). After marking regions containing no relevant information (b), they can be compressed and free additional space for enlarging the remaining important regions (c).

To prevent marking too small regions as unimportant – possibly even smaller than the gap it will be replaced with – we extend all bounding boxes with a large margin below and above, as shown in Figure 3.11. After marking all regions as described before, the height of the important regions is reduced again to get tighter bounds, but still include small uninteresting regions in between.

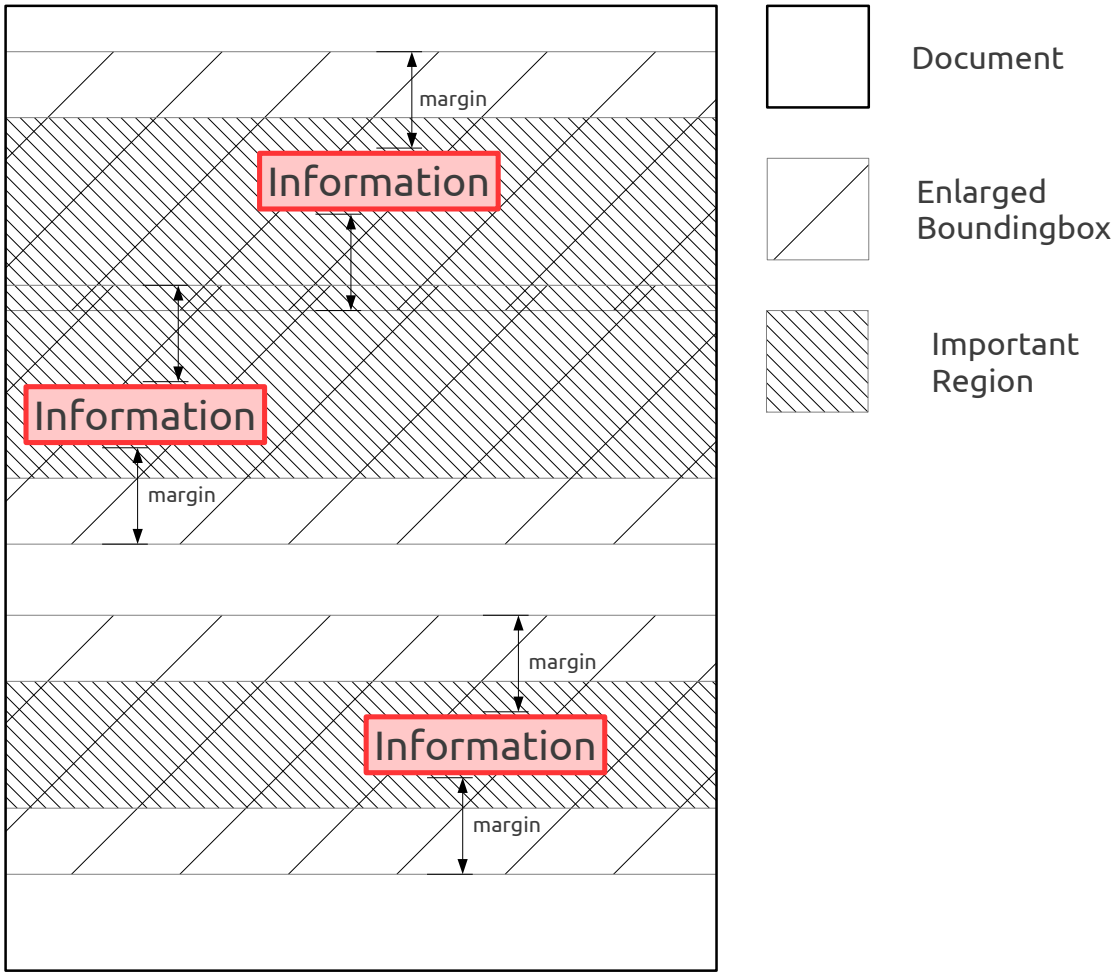


Figure 3.11: Partitioning a document containing three instances of the search identifier. The enlarged bounding boxes are first intersected and afterwards made smaller by reducing the previously introduced margins. Finally the resulting two regions are marked as important.

Chapter 4

Design and Implementation

The software prototype of the presented visual links system consists of multiple components working together in a client-server fashion. A central management application communicates with registered applications via a special *Visual Link Protocol (VLP)*, processes the collected data and draws an according visualization on top of the desktop.

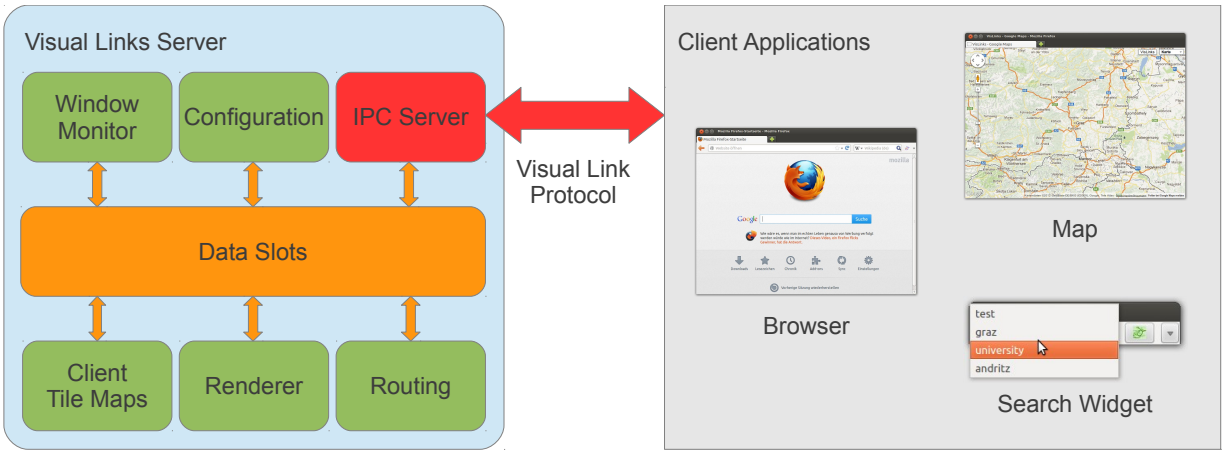


Figure 4.1: Primary components of the visual links system. The visual links server communicates with different client applications and creates visual links according to the received data.

4.1 Visual Links Server

The central server application is written in the C++[1] programming language using the Qt framework[2], which allows the application to support various platforms[3]. Multiple components are communicating via a generic data exchange system using so called *Data Slots*. All components are initially completely independent of each other and are connected with data slots during the initialization phase of the visual links server.

[1] `http://isocpp.org/`
[2] `http://qt-project.org/`
[3] Linux and Windows have been tested

4.1.1 Inter Process Communication

Serving as a gateway between the visual links server and its client applications, the `IPCServer` is a crucial part of the system. Using a special protocol, as described in more detail in Section 4.2, textual as well as binary data is exchanged between the server and its clients. Upon initiating a new routing process – usually through a message from a client – the server requests information related to the received search identifier from all registered clients. The received data – locations and shapes of regions of interest, as well as images containing snapshots of hidden areas – is processed by the `IPCServer` and stored in data slots to be further processed by other components.

Clients are also able to request and change settings of the visual links server or abort existing routes. Upon changes inside their applications content, they should notify the server about it to initiate updating the affected parts of the visualization.

4.1.2 Routing

After the `IPCServer` has collected all available data from each connected client, the `Routing` component uses this data to connect the separate highlighted regions with visual links as discussed in Section 3.1. Additionally to the linked regions and window geometries, periodically the contents on the screen are captured into an image with all links removed. This image can be used by routing components creating context sensitive links like for example described in [SWS+11].

Our prototype though just includes a simple `CPURouting` component, which performs all calculations inside the computers main processing unit. Regions are connect with straight lines, without trying to avoid important content. Links are bundled, as described in Section 3.1.2, to reduce visual clutter.

4.1.3 Renderer

Using the data created by the routing component – vertices connected with straight lines – the `GLRenderer` draws extruded lines and region highlights using the OpenGL[4] API. Afterwards the whole output is blurred using a shader written in the OpenGL Shading Language (GLSL). This leads to smooth edges and corners, nicely blended over the desktop. For platforms, as for example Windows, which do not supporting drawing on top of the desktop using alpha-blending, before drawing the blurred output on top of the desktop, it is blended with an image of the desktop using another GLSL shader to imitate real alpha-blending.

If a preview pop-up (see Section 3.2.2.1) should be shown, the pop-up frame is drawn using standard OpenGL primitives. After applying a Gaussian blur, the preview is drawn using the according tiles of the *Hierarchic Tile Map*, which will be discussed in the next section. Highlights are added using simple lines, as shown for example in Figure 3.9.

[4] `http://www.opengl.org/`

4.1.4 Client Hierarchic Tile Map

For rendering a preview pop-up the respective client has to send an image of its contents to the server. To allow inspecting the preview at different levels of zoom, a high resolution image is needed. Creating and transferring such large images would be very slow and inefficient, as the user probably will not inspect the whole region at the highest available zoom level.

To accommodate for this we use a *Hierarchic Tile Map*, where each level consists of a single or multiple tiles which add up to a full preview image of the client applications region at a specific zoom level. Using different resolution images for the individual zoom levels allows creating tiles in a resolution sufficient for a single level of zoom. As the user zooms into the preview or moves the viewport around, missing tiles are requested asynchronously from the corresponding client application.

Because of rendering and loading only relevant parts on demand, much smaller images are created and subsequently far less data has to be transfered. This also improves the responsiveness of the visualization, as it allows displaying parts of the preview while waiting for missing tiles being received.

4.1.5 Window Monitor

While working in a desktop environment the arrangement of opened windows can change at any time (*cf.* Challenge 1 in Section 1.2). As this possibly also affects the position and occlusion of regions, we need to check for such changes. Current operating systems usually do not allow receiving notifications for changes in windows of other applications. To work around this limitation the `WindowMonitor` periodically request a list of all opened windows, including its geometry and stacking order, and compares them with the stored previous state. If there are any changes detected, it notifies the `IPCServer` such that it can perform the required action, for example trigger recreating routes for active visual links.

4.1.6 Configuration

The used routing algorithm, color scheme, search history and other settings which should be persistent throughout multiple runs of the visual links system, are stored in files on the hard disk. Using TinyXML[5], a library for handling Extensible Markup Language (XML) encoded files, the `XMLConfig` component is responsible for saving and restoring all configuration values.

As modern operating systems are multi-user systems, using at least two configuration files – one stored in a location readable for all users, and further ones accessible to each user – allows providing global default settings, while at the same time allowing each user to override settings persistent across multiple sessions.

[5] `http://www.grinninglizard.com/tinyxml/`

4.2 Visual Link Protocol

As not all data needed by the server can be gathered through monitoring application windows the `IPCServer` (*cf.* Section 4.1.1) is responsible for exchanging data between the different client applications and the core system. It provides a TCP server and accepts incoming WebSocket [FM11] connections, which are used to exchange basic properties about the client, like the size of its scrollable region, and requesting locations of information inside the clients window. As the WebSocket protocol is a bidirectional protocol a client itself can also request data and settings from the server or trigger a linking process for a new search id.

Messages exchanged between the Visual Links Server and its clients are transfered using *VLP*. The VLP is a simple, text based data exchange format using JavaScript Object Notation (JSON) [Cro06] encoded objects. Such an object consists of keys and associated values or list of values.

Every message object consists of a mandatory field `'task'`, which identifies the type of message being transfered, and depending on the actual message type, several other fields. The following example shows a message used to initiate a linking process:

```
{
  'task': 'INITIATE',
  'id': 'search identifier',
  'stamp': 123456789,
  'regions': [
              [[12, 34], [112, 34], [112, 64], [12, 64]],
              [[40, 100], [100, 200], [10, 200]]
            ],
  'scroll-region': [0, 0, 780, 550]
}
```

Listing 4.1: VLP sample message

In the subsequent sections we will describe all messages specified by VLP.

REGISTER

The first task a client has to do after connecting to the Visual Links Server is sending a registration message. It has to contain the coordinates of a position (`'pos'`) inside the clients window and the dimensions of the applications content viewport (`'region'`) excluding any window decoration and menus.

Sending a visible position inside the clients window allows to identify the window in the window stack, by checking the visible window at that location. This information is needed by the window monitor described in Section 4.1.5.

The bounding box of the content region is used for determining whether a received region of interest is visible or hidden due to being scrolled outside the viewport.

```
{
  'task': 'REGISTER',
  'name': 'Optional Name',
  'pos': [px, py],
  'region': [x, y, width, height]
}
```

Listing 4.2: VLP message: register an application

RESIZE

Whenever the viewport geometry of an application changes, the client should notify the server using a resize message. The `'region'` parameter should contain the same data as used withing the REGISTER message.

```
{
  'task': 'RESIZE',
  'region': [x, y, width, height]
}
```

Listing 4.3: VLP message: report application resized

INITIATE

Initiates a new routing and linking process by sending the identifier (`'id'`) to search to the server. Locations inside the clients region can be added in the optional `'regions'` field.

If the document is scrollable the dimensions and offset of the whole documents region are included in the `'scroll-region'` field.

The following example shows a message to initiate a search for the string 'test' with two occurrences bounded by one rectangular and one triangular region:

```
{
  'task': 'INITIATE',
  'id': 'test',
  'stamp': 123456789,
  'regions': [
              [[12, 34], [112, 34], [112, 64], [12, 64]],
              [[40, 100], [100, 200], [10, 200]]
            ],
  'scroll-region': [x-offset, y-offset, width, height]
}
```

Listing 4.4: VLP message: initiate routing

REQUEST

After the client has issued an INITIATE to the server, the server sends a REQUEST message to all clients with the new search id (`'id'`) and timestamp (`'stamp'`). Every client should send back a FOUND message with all occurrences of the requested id inside their region.

The client which sent the corresponding INITIATE also gets a REQUEST. If matching regions have already been included in the INITIATE, this message can be ignored.

The following example shows the REQUEST the server would send upon receiving the example from INITIATE:

```
{
  'task': 'REQUEST',
  'id': 'test',
  'stamp': 123456789
}
```

Listing 4.5: VLP message: request regions of interest

FOUND

Upon receiving a REQUEST message a client shall search for occurrences of the requested identifier (`'id'`) within its content region and report all found regions of interest (`'regions'`).

The parameters have the same meaning as used within an INITIATE message.

```
{
  'task': 'FOUND',
  'id': 'test',
  'stamp': 123456789,
  'regions': [
              [[12, 34], [112, 34], [112, 64], [12, 64]],
              [[40, 100], [100, 200], [10, 200]]
            ],
  'scroll-region': [x-offset, y-offset, width, height]
}
```

Listing 4.6: VLP message: report regions of interest

ABORT

If a client sends an ABORT to the server then the server forwards the message to all clients and removes every displayed route for the given link `'id'`.

If all routes should be aborted set id to the empty string and `'stamp'` to -1.

```
{
  'task': 'ABORT',
  'id': 'test',
  'stamp': 123456789
}
```

Listing 4.7: VLP message: stop linking

GET

A client can request currently set configuration values and parameters by issuing a GET request. The `'id'` contains a string identifying the requested parameter.

```
{
  'task': 'GET',
  'id': '/routing'
}
```

Listing 4.8: VLP message: request current routing algorithm

The same mechanism can also be used the other way round by the server to request data from the client. The following command is used to request a single image for a clients tile map as described in Section 4.1.4.

```
{
  'task': 'GET',
  'id': 'preview-tile',
  'size': [256, 256],
  'sections' : [[10, 100], [150, 800], [1200, 2000]],
  'src': {
    'x': 0,
    'y': 0,
    'width': 200,
    'height': 400
  },
  'req_id': 5
}
```

Listing 4.9: VLP message: request tile image from a client

In contrast to all other messages, the client has to answer a request for a tile image with a binary message. The first two bytes denote the message type and request identifier respectively, whereas all following bytes build up the raw image data.

GET-FOUND

Upon receiving a GET request the server or client respectively answers with a GET-FOUND message containing the requested `'id'` and it's value(s) (`'val'`). In the case binary data needs to be transfered no GET-FOUND is sent but only the binary data instead.

```
{
  'task': 'GET-FOUND',
  'id': '/routing',
  'val': {
    'active:': 'CPURouting',
    'available': [ ['GPURouting', 1],
                   ['CPURouting', 1] ]
  }
}
```

Listing 4.10: VLP message: answer indicating current routing algorithm

SET

Setting a server configuration value can be requested by a client through sending a SET request to the server. The parameter named `'id'` will be set to the given value (`'val'`).

```
{
  'task': 'SET',
  'id': '/routing',
  'val': 'CPURouting'
}
```

Listing 4.11: VLP message: request changing routing algorithm

4.3 Application Integration

All applications on the desktop can potentially cover important information. As discussed in Section 4.1.5 the `WindowMonitor` is able to monitor all windows opened on the desktop. If an application wants to actively participate to the linking process though it has to communicate with the visual links server using VLP as described in Section 4.2. We have created and extended three different applications now being able to be communicate with the visual links server.

4.3.1 Browser Add-On

Mozilla Firefox[6] is an open-source web browser available for most common operating systems. It can be extended with add-ons using the XML User Interface Language (XUL), Cascading Style Sheets (CSS) and the JavaScript [Int11] programming language. Recent versions of Firefox have integrated a WebSocket API which allows our extension to easily communicate with the visual links server. As shown in Figure 4.2 a button is added to the browsers toolbar which has multiple functions. The color indicates the status of the connection to the visual links server. If no connection has been established, pressing the button creates a new connection, otherwise the current mouse selection is sent as new linking request to the visual links server. Using the drop-down menu active links can be aborted and several settings be changed.

[6] `http://www.mozilla.org/`

Web browsers usually show HyperText Markup Language (HTML) [7] encoded documents. The structure of such HTML documents is specified by the Document Object Model (DOM) [8]. Using the JavaScript implementation of the specified interface we scan through the whole document and calculate bounding boxes for all elements containing a specific search identifier. The bounding boxes of all found occurrences, visible as well as outside the current viewport, are sent back to the server for further processing.

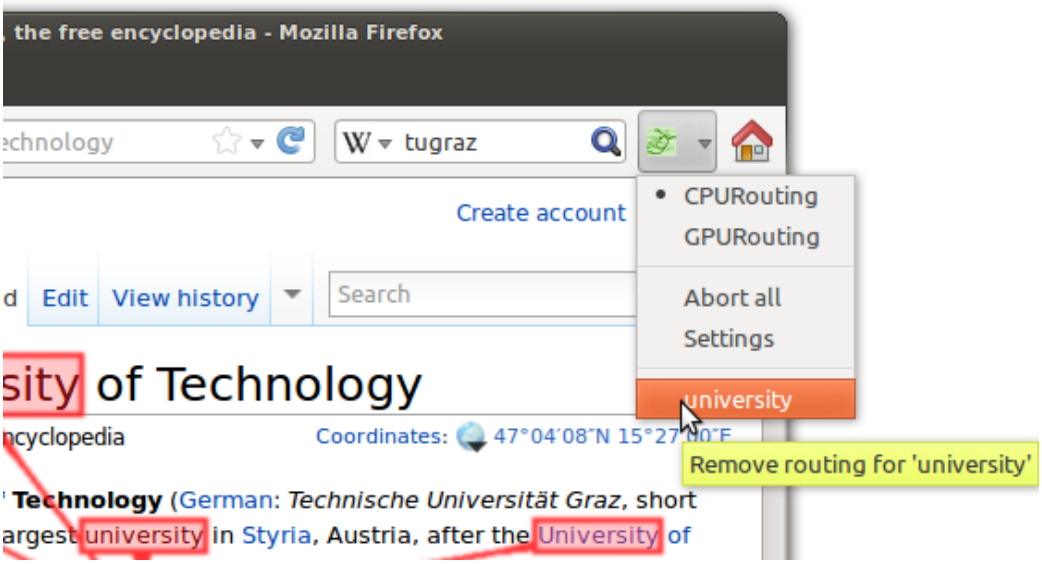


Figure 4.2: Mozilla Firefox with the Visual Links Add-on activated. Currently a linking process for ”university” is active. Upon clicking on the according entry in the linking menu the linking process is aborted.

A search process is triggered either upon receiving a REQUEST message (see Section 4.2) from the visual links server or if the user marks a word or phrase and clicks on the linking button inside the toolbar. Also if the user scrolls or resizes the browser window the search processes is performed again, as it potentially changes the locations of the found regions.

Using the linking menu it is also possible to open a settings dialog. It can be used to change for example the behavior upon starting a new linking process. Either just a new link tree is added or all existing linking process are aborted before starting the new one. Another setting is the algorithm used for routing the links. If available, instead of the `CPURouting` as described in Section 4.1.2 for example a GPU based, context-preserving algorithm could be selected.

[7] `www.w3.org/html/`
[8] `http://www.w3.org/DOM/`

4.3.2 Google Maps Mash-up

Showing geographic locations is not possible by just matching textual data. Using Google Maps we have created a mash-up, a local web site embedding a Google Map instance. With the Google Maps JavaScript API [9] it is possible to search for geographic locations by a search string and retrieve the according screen coordinates on the shown map. The reverse task – retrieving the name of nearby locations for a given screen position – is also supported by the API, which we have used for showing a context menu upon clicking onto the map. As shown in Figure 4.3 this menu can be used to create new links to locations around the cursor.

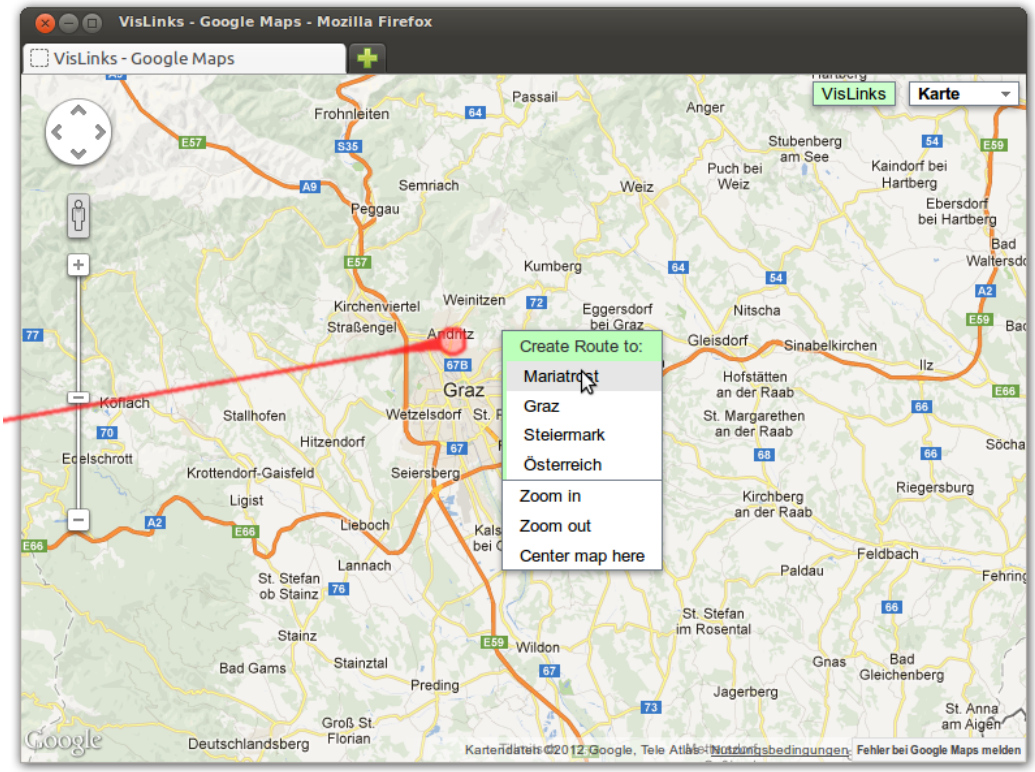


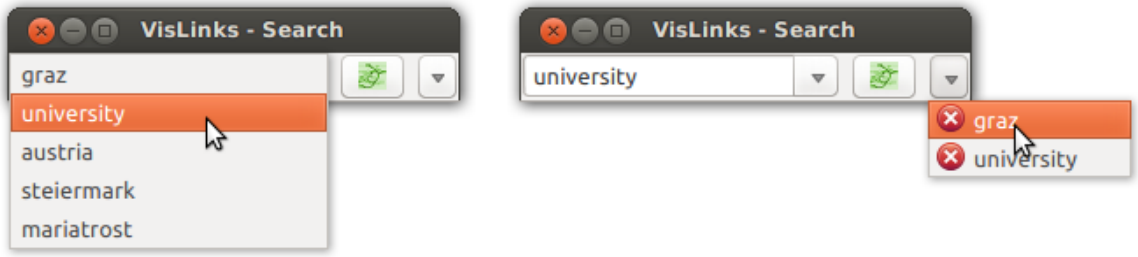
Figure 4.3: Google Maps Mash-up. A link is drawn to the location of Andritz, a district in the city of Graz. The context menu shows that the mouse has been clicked next to the district of Mariatrost. Using the context menu the user can create a new link to the selected location or alternatively to a higher-order administrative level, like city, state or country.

Similar to the browser add-on described in the previous Section 4.3.1, the button labeled ”VisLinks” shows the status of the connection with the visual links server and is used to create a new connection. Again we use the JavaScript WebSocket API for communicating with the visual links server.

[9] `https://developers.google.com/maps/documentation/javascript/tutorial`

4.3.3 Search Widget

The search widget is a small window allowing the user to create new or abort existing linking processes. A textbox allows initiating a new linking process by either entering a new search term or selecting from a list of previously used words (see Figure 4.4(a)).



(a) History drop-down

(b) Abort active routes

Figure 4.4: The search widget allows initiating new linking processes (a) as well as aborting existing ones (b).

A drop-down list, as shown in Figure 4.4(b), provides a list of currently active links. Clicking on an entry in this list sends a request to the visual links server to remove all links for the according search term.

Implemented using the C++ programming language and the Qt framework, the search widget uses the same library as the core visual links server for creating WebSocket connections. Settings and other data, like for example the search history, are handled by the core visual links system and exchanged using the VLP SET and GET commands as described in Section 4.2.

Chapter 5

Results

This chapter discusses the outcome of our work. First we show the implemented methods within different usage scenarios. Afterwards we will analyze the performance of our prototype to show it is able to provide feedback to the user within reasonable time.

5.1 Usage Scenarios

The implementation of our methods discussed in Chapter 3 can be used in a variety of scenarios. It provides an enhanced search and navigation functionality usable inside single applications, but unveils its full potential while interacting with multiple applications at the same time.

5.1.1 Single Window

Figure 5.1: Using the built-in search function of Mozilla Firefox to find information about universities in Graz. One single region is highlighted with a green background whereas no indication for any further region is present.

For searching information inside single windows of applications like web browsers or text editors, often a simple search functionality is provided. Initiated by either just typing or pressing a keyboard shortcut, the viewport jumps to the first matching word. Sometimes also all occurrences of the search term are highlighted (see Figure 5.1) – an in-place F+C technique as described in Section 2.1.4. By scrolling through the document all occurrences of the search term can be inspected. Alternatively by pressing a key or button one can jump to the location of the next occurrence. This can be a tedious process, especially if the number of elements found is large. For reducing the mental load of this task and at the same time increasing the efficiency, visual links can be used.

Visual Links

Figure 5.2: Using the same setup as in Figure 5.1, but using visual links to connect search results instead of the built-in search function. Again a single region is highlighted, but additionally a visual link guides the user towards the 74 hidden regions further down inside the web site. This indication strongly motivates the user to scroll down where he can expect to find more occurrences of the requested search term.

As discussed in Section 2.2.3, motivated by the Gestalt law of connectedness, *visual links* provide strong visual guidance. Using visual links to search for information – for example inside a web browser as shown in Figure 5.2 – helps the user efficiently navigating to every highlighted region (*cf.* Steinberger *et al.* [SWS+11]). The user only needs to follow the visual links towards all found occurrences. If a region is not connected with a visual link it is not considered relevant. This eliminates the need for scanning through

the whole document searching for highlighted regions, as needed with most built-in search functions.

Preview Pop-Up

In Section 3.2.2 we have identified documents or web pages larger than the viewports they are displayed withing, as cause for regions being hidden. By scrolling and zooming the user can still navigate through the whole document and search for such regions. But on the one hand, without exploring the whole content it is not possible to realize if important information is available, and on the other hand, for searching information it is cumbersome to only use zooming and scrolling.

Our visualization technique uses arrows pointing into the directions of such hidden regions to indicate their existence to the user. At the bottom of Figure 5.2 such an arrow is shown. Additionally we have introduced *preview pop-ups* (see Section 3.2.2.1), which shows an outline of the whole document to the user on demand.

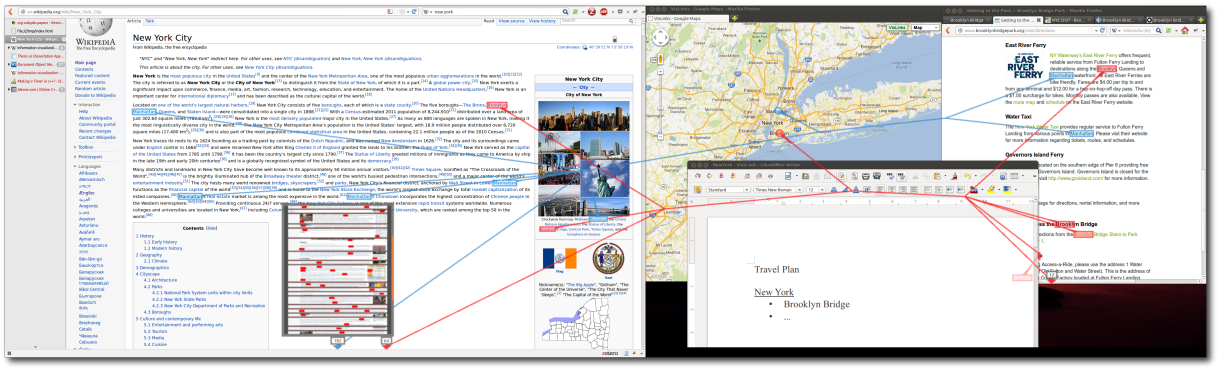
Figure 5.3: The preview pop-up shows a compressed view of the whole document with all regions of interest being highlighted. Darker horizontal bars indicate unrelated regions being filtered, resulting in more space available for drawing the remaining parts of the document. In this image a bit below the center of the preview pop-up a larger region containing many highlights can be seen – likely a region containing lots of relevant information.

Regions containing no relevant information are not rendered, yielding space to enlarge regions of interest (see Figure 5.3). Using a compressed view of the document allows for a fast identification of regions containing lots of information, without being disturbed by

unrelated content. In contrast to simply zooming out of the document it is possible to show the relevant information at a higher zoom level but still using the same amount of screen space.

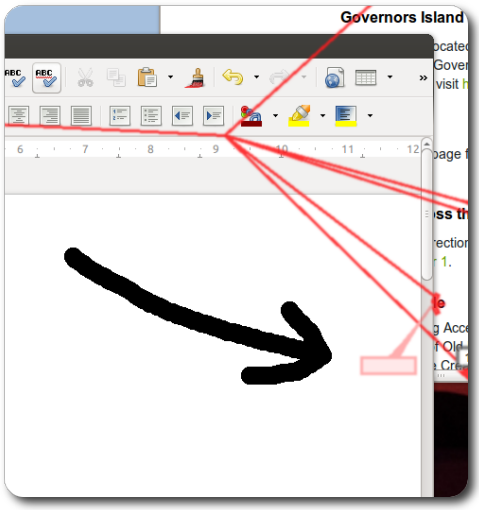
Upon first opening a preview pop-up a scaled image of the whole document is shown. Using the mouse or keyboard shortcuts the user can zoom into the preview and drag it around. After finding an interesting region, with a mouse click the document is scrolled to the according position and the preview pop-up is hidden again.

5.1.2 Multiple Windows



(a) Use-Case Multiple Applications - Traveling to New York

(b) Outside Regions - Preview Pop-up



(c) Covered Region

Figure 5.4: Using visual links to connect related information shown in two browser windows and one map mash-up (a). Information is hidden outside the viewports of the web browsers as well as covered by a simultaneously opened instance of a word processor. Two different search terms are connected at the same time, with one covered region (c) (*cf.* Section 3.2.1) and several regions outside their according viewports with a preview pop-up opened, showing a compressed view of the whole document (b).

In contrast to the default search functionality of many applications, our visual links system uses a client-server architecture (see Chapter 4). This allows creating links not only inside a single application but also between multiple applications. Figure 5.4 shows a possible use-case for such a multi-application visual links setup.

Covered Regions

Most of the time more than one application is opened at once. As soon as this happens regions are prone to being covered by other windows. As described in Section 3.2.1, such *covered regions* are rendered using a transparent color providing the user a see-through view through the covering windows to the hidden regions (see Figure 5.4).

Upon moving the mouse over a covered region, a colored frame is drawn to depict the window the region belongs to (see Figure 5.5). With a click the according window is moved on top of all open windows to allow the user inspecting the now visible region. This requires much less input by the user than for example cycling through all opened windows until finding the correct window.

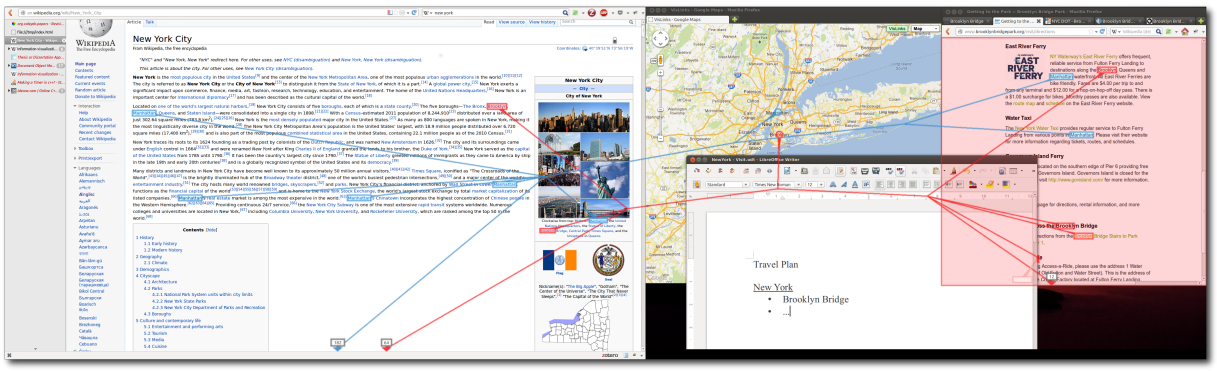


Figure 5.5: Using the same setup as in Figure 5.4(a). Hovering above the covered region highlights the frame of the applications viewport – the red rectangle on the right side – the covered region belongs to.

5.2 Performance

In an interactive system performance plays a crucial role. To identify possible performance bottlenecks we have conducted benchmarks for the different parts of our visual links system.

5.2.1 Core System

The core system (see Section 4.1) communicates with client applications, calculates routes for visual links and finally renders them onto the desktop. The diagram in Figure 5.6 shows the times needed by major parts of the core system.

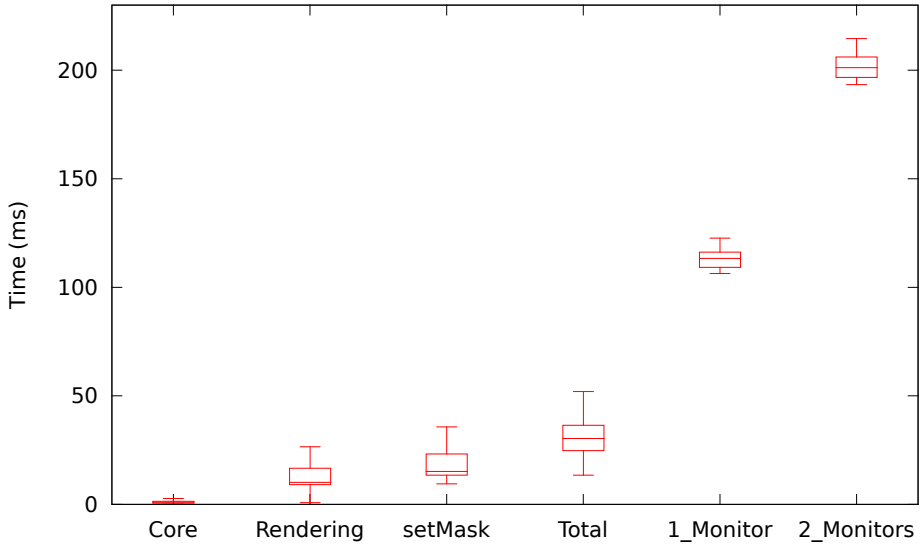


Figure 5.6: Visual Links Server - Benchmark

Times have been measured during multiple test runs on a consumer computer containing an Intel Core i5-3570K quad-core processor with 3.4 GHz clock frequency and a Nvidia GTX 275 graphics card. A single pass of calculations has to be performed every time the links may change upon recalculating. Major reasons for this are moved or resized windows and creating new links or removing existing ones.

With a time of almost always below three milliseconds, the delay caused by the routing algorithm and commands (`Core`) being sent to the graphics card is barely noticeable. Afterwards the system has to wait for the graphics card to finish it works (`Rendering`) which leads to a delay with its median value at about 10 milliseconds. Subsequently the rendered links have to be read back from the graphics card to the processor and a mask, marking all pixels not containing any visible part of the visualization, has to be applied (`setMask`) to the window of the visual links system. Due to the large amount of data being transfered through the computers buses this is the slowest part of the core system, taking a median time of about 16 milliseconds. Together all parts of the core system (`Total`) need about 35 milliseconds on average.

If using a context-aware routing algorithm or running on platforms not supporting direct blending with the desktop, an image of the base representation without any links is required. To achieve this an image of the screen is capture which unfortunately also contains the links currently displayed. To get rid of this links upon rendering they are always drawn with reduced opacity. This allows to remove the links from the image by subtracting the color values of each rendered pixel from the according pixel in the screen capture and scaling it up to the full color range. About one bit of precision is lost on each color channel, which is barely noticeable to the unaided eye and context aware routing algorithms. The time needed to calculate a clean screen-shot is highly dependent on the screen resolution and is shown in the diagram in Figure 5.6 for a single monitor (`1_Monitor`) with a resolution of 1920x1080 pixels, and two monitors (`2_Monitors`) with a combined reso-

lution of 3600x1080 pixels. As the diagram shows this adds a significant delay of about 120 or 200 milliseconds respectively to the time already needed for the core system. As the delay of the context aware routing algorithm is not included, further delays have to be expected.

5.2.2 Client Applications

Before the core system is able to create any routes a client application needs to search for relevant regions and send their locations to the core. Initiating a new routing process using the search widget, as described in Section 4.3.3, only requires entering a string. As this will not lead to any user noticeable delay we have only performed benchmarks for the browser add-on and the map mash-up.

5.2.2.1 Browser Add-on

Figure 5.7 shows some performance measurements for the Mozilla Firefox extension described in Section 4.3.1. It shows the time the add-on needs for finding all occurrences of a search term and calculating their bounding boxes. As can be seen the time directly depends on the number of occurrences a document contains. This is due to the fact that calculating the bounding box of an element requires modifying the DOM tree, a potentially slow operation. The exact delay is also strongly affected by the complexity of the page, which is not necessarily directly related to the file size of the document.

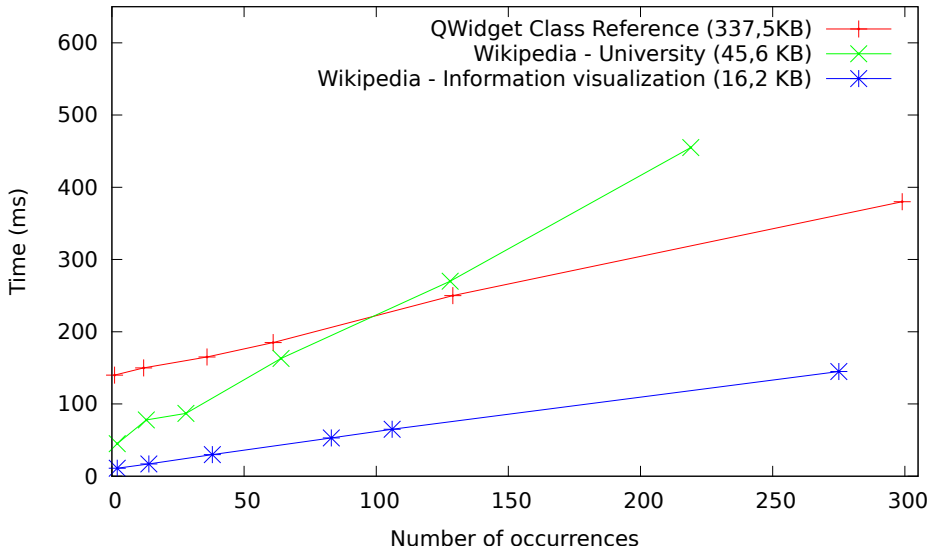


Figure 5.7: Firefox Plug-in Performance: The time it takes the Firefox plug-in to get all bounding boxes of a single search term depends approximately linear on the number of occurrences. (see A.1, A.2 and A.3 for the used search terms and their according times).

For processing 100 occurrences of a single term, depending on the complexity of the website, between 50ms and 220ms have been measured. In the worst case this is about the same delay as introduced by the whole core system creating cleaned screenshots of two monitors. As most of the time probably few occurrences are present, the average delay is much lower and hardly noticeable.

Preview Tile Rendering

On demand of the user a preview pop-up is shown (see Section 3.2.2.1). To show the contents of the pop-up the core system requests image tiles from its client applications. Currently only the browser add-on is able to create such tiles. Upon receiving a tile request (*cf.* Section 4.2) the according part of the web site is rendered into a memory buffer and send back to the visual links server. The diagram in Figure 5.8 shows the total delay from requesting a tile until receiving the according image.

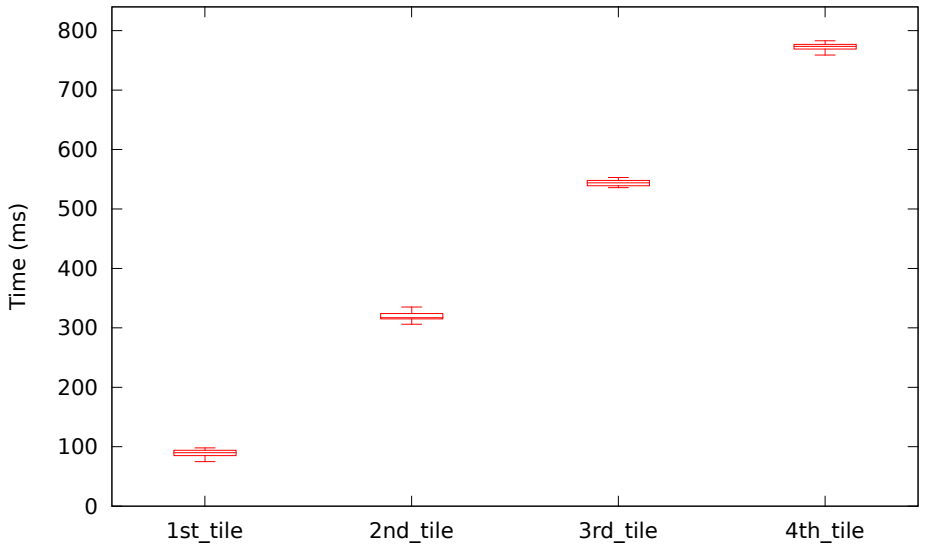


Figure 5.8: Tile Rendering Performance: The Firefox plug-in can receive up to four tile requests at the same time. The plot of the times needed to handle each of the four requests show clearly that they are handled sequentially.

In the worst case up to four tiles are requested at the same time. The first requested tile (`1st_tile`) is always the fastest because all subsequent request can only be completed after the previous ones have been completely processed. Once a tile has been received it is immediately send to the graphics card to be shown to the user. After receiving the first tile an additional delay can be observed, caused by the core system uploading the image to the graphics card and updating the preview popup.

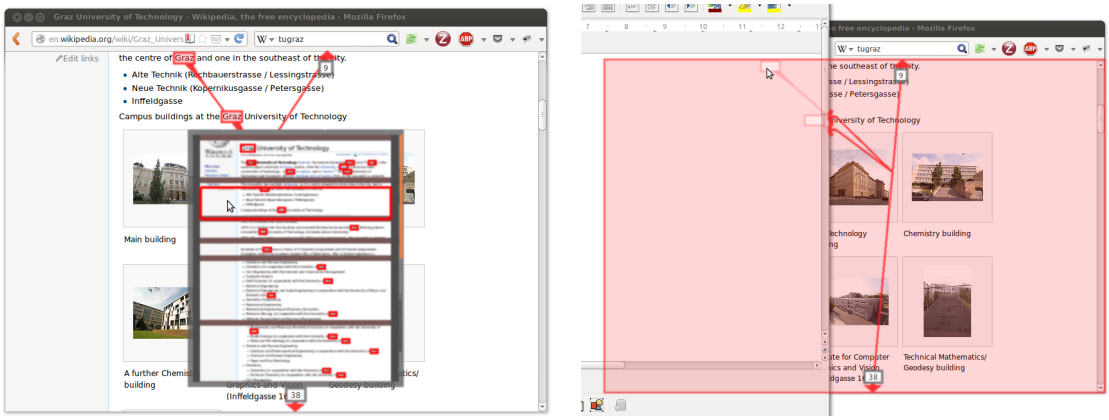
5.2.2.2 Google Maps Mash-up

The Google Maps mash-up, as discussed in Section 4.3.2, uses an API provided by Google to lookup geographic locations for textual search identifiers. The delay highly depends on the quality of the connection to the servers of Google. For us it always took about 80 milliseconds to retrieve a result. If subsequently looking up an already known location, even after panning or zooming the map, results are cached which results in a reduction of the delay to a total time needed of less than three milliseconds.

Chapter 6

Conclusions and Future Work

Visual links create strong visual connections between related content visible on the screen. We have extended this idea to also create guidance towards hidden content. Integrated in a visual links system two new visualization methods create visual cues towards content covered by other windows as well as content outside the visible viewport of applications. Preview pop-ups allow efficiently exploring content outside visible areas by providing an content sensitive, space conserving, compressed preview of a whole documents virtual area (See Figure 6.1(a)), whereas a see-through interface using transparent links with a click-to-show functionality allows for a fast navigation to covered content (See Figure 6.1(b)).



(a) Preview Pop-up (b) Covered Region

Figure 6.1: Using our visual links system to connect related information. A preview pop-up (a) shows a compressed view of content hidden outside the viewport of an application. If a region is covered by a window, upon moving the mouse over the visualization of the covered region, the vieport of the according application is depicted by drawing its outline (b).

A prototype of the proposed techniques is available for multiple platforms. Provided plug-ins allow using a web browser and a searchable map mash-up with the visual links system. A search widget is available to initiate linking processes between content matching entered search strings inside all connected applications.

Future Work

Once reaching a certain number of regions connected at the same time, the visualization gets cluttered and distracts from actual content. To reduce this unintended effects of the visual links more sophisticated bundling algorithms and context aware routing algorithms could be used – for example *Context-preserving visual links* as developed by Steinberger *et al.* [SWS+11].

Another problem worth further investigation are overlapping regions. If regions – independent whether they belong to the same or different search terms – inside multiple windows overlap (See Figure 6.2), the coinciding regions make it hard to identify to which windows they belong to.

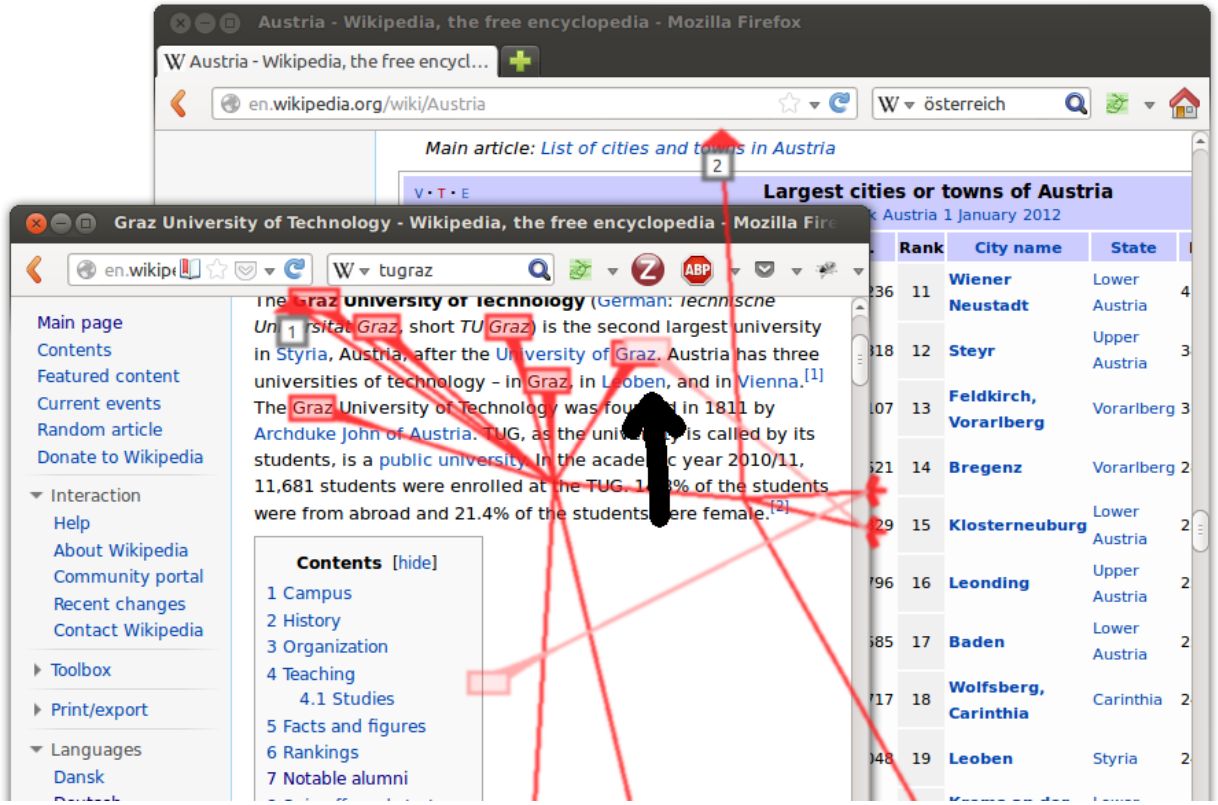


Figure 6.2: Overlapping regions (see arrow) can be hard to distinguish. Also it can make it impossible to directly navigate to the window the lower regions belongs to, as the mouse over event is captured by the upper region.

Acknowledgements

First, I want to thank my supervisors Markus Steinberger, Marc Streit and Alexander Lex for their continuous support and feedback over the course of this thesis. I also want to thank Dieter Schmalstieg for enabling me to work at the Institute for Computer Graphics and Vision.

Furthermore, I want to thank my family and friends, who have enriched my life in many ways.

Finally, I want to specially thank my parents for supporting me throughout my studies and every other part of my whole life.

Appendix A

Performance Data

Search term	# of occurrences	Search time
QWidget	299	380ms
bool	129	250ms
class	61	185ms
QEvent	36	165ms
QFont	12	150ms
#include	1	140ms

Table A.1: Firefox http://doc.qt.digia.com/4.7-snapshot/qwidget.html (337,5KB)

Search term	# of occurrences	Search time
university	219	455ms
human	128	270ms
by	64	163ms
student	28	87ms
learn	13	78ms
bachelor	2	45ms

Table A.2: Firefox http://en.wikipedia.org/wiki/University (45,6KB)

Search term	# of occurrences	Search time
is	275	145ms
visual	106	65ms
visualization	83	53ms
information	38	30ms
human	14	17ms
numerical	2	11ms

Table A.3: Firefox http://en.wikipedia.org/wiki/Information_visualization (16,2 KB)

Acronyms

CSS Cascading Style Sheets. 41

DOM Document Object Model. 42, 51

F+C Focus+Context. 11, 12, 14, 23, 46

GLSL OpenGL Shading Language. 36

HTML HyperText Markup Language. 42

JSON JavaScript Object Notation. 38

VLP Visual Link Protocol. 35, 38, 41, 44

XML Extensible Markup Language. 37

XUL XML User Interface Language. 41

List of Figures

List of Tables

Listings

Bibliography

[AS07] Aleks Aris and Ben Shneiderman. Designing semantic substrates for visual network exploration. *Information Visualization*, 6(4):281–300, December 2007.

[BB09] Xiaojun Bi and Ravin Balakrishnan. Comparing usage of a large high-resolution display to single or dual desktop displays for daily work. page 1005. ACM Press, 2009.

[BCG06] Stefano Burigat, Luca Chittaro, and Silvia Gabrielli. Visualizing locations of off-screen objects on mobile devices. page 239. ACM Press, 2006.

[BCR+03] Patrick Baudisch, Edward Cutrell, Dan Robbins, Mary Czerwinski, Peter Tandler, Peter T, Benjamin Bederson, and Alex Zierlinger. Drag-and-pop and drag-and-pick: techniques for accessing remote screen content on touch- and pen-operated systems. pages 57–64, 2003.

[BDB06] Anastasia Bezerianos, Pierre Dragicevic, and Ravin Balakrishnan. Mnemonic rendering: an image-based approach for exposing hidden changes in dynamic displays. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, UIST '06, page 159–168, New York, NY, USA, 2006. ACM.

[BE96] Thomas Ball and Stephen G Eick. Software visualization in the large. *Computer*, 29(4):33–43, 1996.

[Bec87] Richard A. Becker. Dynamic graphics for data analysis. *Statistical Science*, 2(4):355–383, November 1987.

[BR03] Patrick Baudisch and Ruth Rosenholtz. Halo: a technique for visualizing off-screen objects. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '03, pages 481–488, New York, NY, USA, 2003. ACM.

[BSP+93] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and magic lenses: the see-through interface. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 73–80, New York, NY, USA, 1993. ACM Press.

[CC07] C. Collins and S. Carpendale. Vislink: Revealing relationships amongst visualizations. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1192–1199, nov.-dec. 2007.

[CKB09] Andy Cockburn, Amy Karlson, and Benjamin B. Bederson. A review of overview+detail, zooming, and focus+context interfaces. *ACM Comput. Surv.*, 41(1):2:1–2:31, January 2009.

[CPC09] Christopher Collins, Gerald Penn, and Sheelagh Carpendale. Bubble sets: Re-

vealing set relations with isocontours over existing visualizations. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '09)*, 15(6):1009–1016, 2009.

[Cro06] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational), July 2006.

[CZQ+08] Weiwei Cui, Hong Zhou, Huamin Qu, Pak Chung Wong, and Xiaoming Li. Geometry-based edge clustering for graph visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1277–1284, December 2008.

[DEGM04] Matthew Dickerson, David Eppstein, Michael T. Goodrich, and Jeremy Yu Meng. Confluent drawings: Visualizing non-planar diagrams in a planar way. pages 1–12, January 2004.

[ED07] Geoffrey Ellis and Alan Dix. A taxonomy of clutter reduction for information visualisation. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1216–1223, 2007.

[EHJS01] James Robinson Eagan, Mary Jean Harrold, James Arthur Jones, and John T. Stasko. Visually encoding program test information to find faults in software. http://smartech.gatech.edu/handle/1853/3324, 2001.

[ESS92] S.C. Eick, J.L. Steffen, and Jr. Sumner, E.E. Seesoft-a tool for visualizing line oriented software statistics. *IEEE Transactions on Software Engineering*, 18(11):957 –968, November 1992.

[ET08] N. Elmqvist and P. Tsigas. A taxonomy of 3D occlusion management for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(5):1095–1109, September 2008.

[FM11] I. Fette and A. Melnikov. The WebSocket Protocol. RFC 6455 (Proposed Standard), December 2011.

[Fur81] George W. Furnas. The FISHEYE view: A new look at structured files. *Bell Laboratories Technical Memorandum*, (81-11221-9), October 1981.

[Fur86] George W. Furnas. Generalized fisheye views. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '86)*, pages 16–23, New York, NY, USA, 1986. ACM Press.

[GBGI08] Sean Gustafson, Patrick Baudisch, Carl Gutwin, and Pourang Irani. Wedge. page 787. ACM Press, 2008.

[GHNS11] E.R. Gansner, Yifan Hu, S. North, and C. Scheidegger. Multilevel agglomerative edge bundling for visualizing large graphs. In *Pacific Visualization Symposium (PacificVis), 2011 IEEE*, pages 187–194, march 2011.

[HF01] Kasper Hornbaek and Erik Frøkjær. Reading of electronic documents: The usability of linear, fisheye, and overview+detail interfaces. In *In CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 293–300. ACM Press, 2001.

[Hol06] Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer*

Graphics (InfoVis '06), 12(5):741–748, 2006.

[HS04] Dugald Ralph Hutchings and John Stasko. Revisiting display space management: understanding current practice to inform next-generation design. pages 127–134. Canadian Human-Computer Communications Society, May 2004.

[HVW09] Danny Holten and Jarke J. Van Wijk. Force-directed edge bundling for graph visualization. *Computer Graphics Forum*, 28(3):983–990, 2009.

[Int11] Ecma International. *Standard ECMA-262: ECMAScript Language Specification*. ECMA (European Association for Standardizing Information and Communication Systems), Geneva, Switzerland, 5.1 edition, 2011.

[JS91] B. Johnson and B. Shneiderman. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *Proceedings of the IEEE Conference on Visualization (Vis '91)*, page 284–291, 1991.

[KF09] J. Krüger and T. Fogal. Focus and context-visualization without the complexity. In Ratko Magjarevic, Olaf Dössel, and Wolfgang C. Schlegel, editors, *World Congress on Medical Physics and Biomedical Engineering, September 7 - 12, 2009, Munich, Germany*, volume 25/13, pages 45–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[KHG03] Robert Kosara, Helwig Hauser, and Donna L. Gresh. An interaction view on information visualization. In *State-of-the-Art Proceedings of EUROGRAPHICS 2003 (EG 2003)*, pages 123–137, 2003.

[KLS00] M. Kreuseler, N. Lopez, and H. Schumann. A scalable framework for information visualization. In *IEEE Symposium on Information Visualization, 2000. INFOVIS 2000*, pages 27–36, Salt Lake City, UT, USA, 2000. IEEE Computer Society.

[KMFK05] Azam Khan, Justin Matejka, George Fitzmaurice, and Gordon Kurtenbach. Spotlight: directing users' attention on large displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '05, pages 791–798, New York, NY, USA, 2005. ACM.

[KMH01] Robert Kosara, Silvia Miksch, and Helwig Hauser. Semantic depth of field. *Information Visualization, IEEE Symposium on*, 0:97, 2001.

[KMH+02] Robert Kosara, Silvia Miksch, Helwig Hauser, Johann Schrammel, Verena Giller, and Manfred Tscheligi. Useful properties of semantic depth of field for better f+c visualization. In *Proceedings of the symposium on Data Visualisation 2002*, VISSYM '02, pages 205–210, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[LA94] Y. K. Leung and M. D. Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 1(2):126–160, January 1994.

[LRP95] John Lamping, Ramana Rao, and Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages

401–408, Denver, Colorado, United States, 1995. ACM Press/Addison-Wesley Publishing Co.

[MRC91] Jock D. Mackinlay, George G. Robertson, and Stuart K. Card. The perspective wall: detail and context smoothly integrated. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '91)*, pages 173–176. ACM Press, 1991.

[MW95] Allen R. Martin and Matthew O. Ward. High dimensional brushing for interactive exploration of multivariate data. In *Proceedings of the 6th conference on Visualization '95*, VIS '95, pages 271–, Washington, DC, USA, 1995. IEEE Computer Society.

[NS00] Chris North and Ben Shneiderman. Snap-together visualization: A user interface for coordinating visualizations via relational schemata. In *Proceedings of the ACM Conference on Advanced Visual Interfaces (AVI '00)*, pages 128–135. ACM, 2000.

[PR94] Stephen Palmer and Irvin Rock. Rethinking perceptual organization: the role of uniform connectedness. *Psychonomic Bulletin and Review*, 1(1):29–55, 1994.

[PXY+05] Doantam Phan, Ling Xiao, Ron Yeh, Pat Hanrahan, and Terry Winograd. Flow map layout. In *Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization*, pages 219–224, Washington, DC, USA, 2005. IEEE Computer Society.

[RMC91] George G Robertson, Jock D Mackinlay, and Stuart K Card. Cone trees: animated 3D visualizations of hierarchical information. In *Proceedings of the SIGCHI Conference on Human factors in Computing systems (CHI '91)*, pages 189–194. ACM Press, 1991.

[SA06] B. Shneiderman and A. Aris. Network visualization by semantic substrates. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):733–740, sept.-oct. 2006.

[SB94] Manojit Sarkar and Marc H. Brown. Graphical fisheye views. *Commun. ACM*, 37(12):73–83, 1994.

[SLK+09] Marc Streit, Alexander Lex, Michael Kalkusch, Kurt Zatloukal, and Dieter Schmalstieg. Caleydo: Connecting pathways and gene expression. *Bioinformatics*, 25(20):2760–2761, 2009.

[SSTR93] Manojit Sarkar, Scott S. Snibbe, Oren J. Tversky, and Steven P. Reiss. Stretching the rubber sheet: a metaphor for viewing large layouts on small screens. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST ' 93)*, pages 81–91. ACM Press, 1993.

[Ste76] Saul Steinberg. View of the world from 9th avenue. *The New Yorker*, page 1, March 1976.

[SWS+11] Markus Steinberger, Manuela Waldner, Marc Streit, Alexander Lex, and Dieter Schmalstieg. Context-preserving visual links. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2249 –2258, dec. 2011.

[TAvHS06] C. Tominski, J. Abello, F. van Ham, and H. Schumann. Fisheye tree views and lenses for graph visualization. In *Proc. Tenth International Conference on Information Visualization IV 2006*, page 17–24, July 2006.

[TG80] A M Treisman and G Gelade. A feature-integration theory of attention. *Cognitive Psychology*, 12(1):97–136, 1980.

[VCWP96] John Viega, Matthew J. Conway, George Williams, and Randy Pausch. 3D magic lenses. In *Proceedings of the 9th annual ACM symposium on User interface software and technology*, UIST '96, page 51–58, New York, NY, USA, 1996. ACM.

[VM12] C. Viau and M. J. McGuffin. ConnectedCharts: explicit visualization of relationships between data graphics. *Computer Graphics Forum*, 31(3pt4):1285–1294, 2012.

[War04] Colin Ware. *Information visualization: Perception for design*. Morgan Kaufman, San Francisco CA, second edition, 2004.

[Wer23] Max Wertheimer. Untersuchungen zur Lehre von der Gestalt. II. *Psychologische Forschung*, 4(1):301–350, 1923.

[WPL+10] Manuela Waldner, Werner Puff, Alexander Lex, Marc Streit, and Dieter Schmalstieg. Visual links across applications. In *Proceedings of Graphics Interface 2010*, GI '10, pages 129–136, Toronto, Ont., Canada, Canada, 2010. Canadian Information Processing Society.

[WS11] M. Waldner and D. Schmalstieg. Collaborative information linking: Bridging knowledge gaps between users by linking across applications. In *Pacific Visualization Symposium (PacificVis), 2011 IEEE*, pages 115 –122, march 2011.

[ZK10] Caroline Ziemkiewicz and Robert Kosara. Laws of attraction: From perceptual forces to conceptual similarity. *IEEE Transactions on Visualization and Computer Graphics (InfoVis '10)*, 16(6):1009–1016, 2010.

[ZMG+03] Polle T. Zellweger, Jock D. Mackinlay, Lance Good, Mark Stefik, and Patrick Baudisch. City lights: contextual views in minimal space. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '03, pages 838–839, New York, NY, USA, 2003. ACM.

[ZWSK97] Shumin Zhai, Julie Wright, Ted Selker, and Sabra-Anne Kelin. Graphical means of directing user's attention in the visual interface. In *Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction*, INTERACT '97, pages 59–66, London, UK, UK, 1997. Chapman & Hall, Ltd.